UNIVERSITY OF Southampton

Comparison of numerical methods for propagating the orbit of Apophis

Supervisor: Dr Alexander Wittig
George Galvin, BSc

Submitted for an MSc in Space Systems Engineering

Faculty of Engineering and Physical Sciences
Submitted September 2019

This thesis was submitted for examination in September 2019. It does not necessarily represent the final form of the thesis as deposited in the University after examination.

I, George Galvin, declare that this thesis and the work presented in it are my own and has been generated by me as the result of my own original research. I confirm that:

1. This work was done wholly or mainly while in candidature for a degree at this University;

2. Where any part of this thesis has previously been submitted for any other qualification at this University or any other institution, this has been clearly stated;

3. Where I have consulted the published work of others, this is always clearly attributed;

4. Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work;

5. I have acknowledged all main sources of help;

6. Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself;

7. None of this work has been published before submission.

Abstract

In space, the motion of objects is most easily described by using differential equations. To integrate the forces acting on an object, 'propagating' its position from a known starting point, the complex differential equation describing them is in general impossible to analytically solve. Therefore, *numerical* integration is at the heart of all propagation solutions today. The aim of this Master's project is to study and compare the efficiency of state-of-the art numerical integration methods to the less well-studied Multistep Collocation Methods. In addition, the various choices involved in building the model, and their effect on accuracy and efficiency, will be analysed and discussed. This will be done using the orbit propagation of the near-Earth asteroid (99942) Apophis, and its close encounter with Earth, as a test problem. The novel contribution of this project is to directly compare various MCMs with state-of-the-art methods using the same specific benchmarking problem, leading to much greater clarity on the differences between the methods. The subsequent results and discussion may prove useful for propagating other asteroids and planetary bodies more efficiently.

Contents

List of Figures

List of Tables

Nomenclature

Symbol	Definition
G	Newtonian gravitational constant
c	Speed of light in a vacuum
$\mathbf{y}$	State vector of Apophis
$\mathbf{f}$	Right-hand side (velocity & acceleration)
t	Time
h	Integration time step
t_0	Integration start time
$\mathbf{y}_0$	Initial state vector
a_{ij}	Multistep coefficients
b_{ij}	Runge-Kutta coefficients
c_i	Spacing coefficients
m_i	Mass of body i
$\mathbf{r}_i$	Distance between body i and Solar System Barycenter
β, γ	Post-Newtonian Parameters
au	Astronomical unit

1 Introduction

1.1 The problem

A fundamental problem in orbital mechanics is to predict an object's future trajectory, known as its *ephemeris*, given known initial conditions (position, velocity, and time). Due to Newton's Laws, the change is most easily modelled by a second-order differential equation:

$$\begin{aligned} \mathbf{r}''(t) &= \mathbf{f}(t, \mathbf{r}), \\ \mathbf{r}(t_0) &= \mathbf{r}_0, \\ \mathbf{r}'(t_0) &= \mathbf{v}_0. \end{aligned} \tag{1}$$

Here, $\mathbf{r}$ is the position, $\mathbf{r}_0$, $\mathbf{v}_0$ and t_0 are the initial conditions, and $\mathbf{f}$ represents the net acceleration from all forces acting on it ($\mathbf{f}$ is also referred to as the *right-hand side*). This allows a range of forces to be modelled, if we either know the resultant acceleration directly, or if we know the force's magnitude and the body's mass.

Introducing initial conditions guarantees a unique solution for $\mathbf{r}(t)$ at a future time. This second-order model can be used to propagate a range of objects at different scales, from planets and stars down to spacecraft.

As $\mathbf{f}$ is a function of both $\mathbf{r}$ and t, we typically have a complex non-homogeneous equation. Due to Newton's law of gravitation, $\mathbf{f}$ involves inverse powers of $\mathbf{r}$ which also makes the equation *non-linear*. The particular problem is usually impossible to solve analytically [34]. Therefore, *numerical* methods are typically used instead to find a solution for a given set of time points. This numerical computation is known as *propagating* the body's trajectory.

Numerical propagation can be very time-consuming, especially when the high complexity of Solar System perturbations ([11], chapter 12) are included in the right-hand side, and a high-accuracy result is required over a long time frame. For this reason, the key goal in choosing a numerical method is to maximise *efficiency*, which can be defined as accuracy achieved for a given computational time.

Almost all of the current state-of-the-art numerical methods used to solve differential equations can be divided into two families. These are *multistage* and *multistep* methods, and work by computing partial Taylor expansions of $\mathbf{r}(t)$ - discussed thoroughly in Section 2.3. A newer family of methods, called *Multistep Collocation Methods*, combine the ideas of both families into one algorithm, with the idea of retaining the advantages of both families to achieve higher efficiency. This leads to the primary aim of this project - to generate results that compare the efficiency of Multistep Collocation Methods to that of state-of-the-art integrators.

For each numerical method, precision parameters can be given, which influence both accuracy and computational time. To compare efficiency between methods, a test problem is required, to be first implemented in code, run with each configuration of integrator & precision parameters. I am going to imple-

ment a test problem (right-hand side) as a coded function, and then measure the efficiency of the use of each integrator. The integrators will all have been written already and are not novel to this project.

There are two other obvious ways to decrease computational time. The first is when the propagation is only required to achieve a specific accuracy. In this case, the model may be simplified to remove unnecessary complexity. The issue here is it requires the accuracy cost for each simplification to be known beforehand. This leads to a secondary aim - to review each simplification that can be made, and its quantitative effect on accuracy.

The second way to minimise time is by increasing the efficiency of the model's *implementation*, the way it's encoded into machine-readable form. Another secondary goal of this project is to review the various factors involved in implementing a model, and the most efficient ways to do so.

1.2 Apophis and its fly-bys

Near-Earth Asteroids (NEAs) constantly present a risk to life on Earth [25] with their threat of impact. The monitoring of a dangerous NEA is clearly a highly important use of the modelling techniques above. Additionally, it is one which requires high accuracy - the difference between an impacting and non-impacting NEA may be very small - over long time frames.

(99942) Apophis, originally designated 2004 MN4 [16], is perhaps the most famous NEA. It's well known for its future close approach, or *fly-by*, with Earth on April 13, 2029. In particular, shortly after its 2004 discovery, NEA monitoring systems briefly gave Apophis a 2.7% probability of colliding with Earth during the approach [10] [1]. This was picked up by mainstream media worldwide, and gave Apophis a rating of level 4 on the Torino scale - an impact hazard scale by MIT - the highest rating a NEA has achieved to date. The level of concern was such that possible orbit deflection missions to it were taken very seriously by leading academics [12].

Further observations quickly corrected its trajectory to one with a negligible chance of Earth impact, but with a guaranteed close approach within the geostationary orbit region. Recent estimates give approaches of about 38000km from Earth's center [18].

Figure 1 shows the total magnitude of acceleration experienced by Apophis from 2006 to 2036. (The modelling of this is broken down in section 3). The majority of the time span shows its orbit around the Sun, with acceleration peaks at perihelion and troughs at aphelion. The plot is punctuated by the spike given by the 2029 fly-by, which switches Apophis into an orbit further from the Sun.

Gravitational 'keyholes' are regions of space which, if an asteroid passes through them at exactly the right time, puts the asteroid on a precise trajectory toward a future Earth collision. The 2029 entry into Earth's gravitational field drastically increases the uncertainty region of its post-2029 trajectory. This region includes dozens of keyholes [32], resulting in a 21st century collision unable to be ruled out.

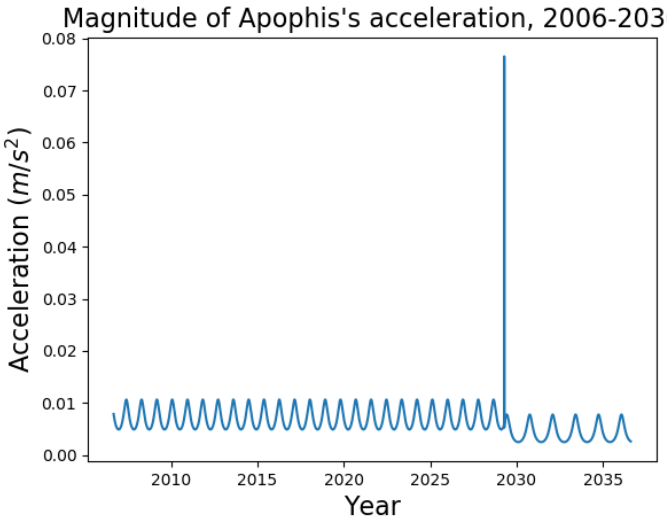

Figure 1: Magnitude of acceleration felt by Apophis, from 2006 to 2036

The current likeliest impact is estimated by NASA's Sentry monitoring system as at April 2068, with an impact probability of 6.7 in a million. However, the cumulative possibility is less than one of a random collision by a yet-undetected object of similar size during this time span.

1.3 Apophis's trajectory as a test problem

To provide a metric for accuracy - to be compared with time - a suitable summary statistic from the test problem is required at the end of each computation. This leads to one motivation for choosing Apophis's ephemeris - the *distance* of closest approach during the fly-by is excellent for this purpose. The quantity is in the 10^8 m range, which is relatively tiny compared to the actual distances being integrated (Apophis's distance from the Solar System Barycenter), which are distances in the 10^{11} m range. This means three things:

- A tiny error in the model's implementation will have a huge impact on the closest approach distance. Therefore, the model can be validated by comparing its closest approach to one in the literature, which is relatively substantial due to the scientific interest in Apophis.
- The more accurate an integrator is, the closer the propagation's closest approach will be to the 'true' value (the value obtained with a very fine step size). This allows a one-dimensional metric for integrator accuracy.
- The effect of changing different model parameters, such as gravitational

equations or ephemerides, can be easily quantified using the closest approach distance.

The second motivation is the *stiffness* of Apophis's ephemeris. A *stiff* differential equation is one whose solution has rapid variation at certain time intervals, while being relatively stable in the rest of the time space ([17], 8.12). Figure 1 shows that Apophis's motion is a clear example of stiffness. Here, the rapid variation interval is the close approach period in 2029, the rest of the time period being benign in comparison.

Stiff equations pose a challenge for numerical integrators, as the fineness of time steps required to stably model the rapid variation periods may be overly redundant when used for the rest of the solution, resulting in huge inefficiencies in computational time. Therefore, the relative efficiency between methods will be more enlightening with this test problem.

For these reasons, the Apophis trajectory model is the test problem used in this report.

1.4 The exact model

In order to test the accuracy of created right-hand side code, it is necessary to find a propagation in the literature that details their exact initial conditions ($\mathbf{y}_0$), right-hand side, and closest approach, replicate it as closely as possible, and verify that the output closest approach from the literature and the replication match.

The most suitable propagation found was in Giorgini(2008) [14], for the following reasons:

- The full initial conditions were given precisely.
- The exact right-hand side was given precisely.
- The distance of closest approach has been given to 3 significant figures in AU. The same right-hand side & initial conditions have also been tested in [18], with the closest approach given in kilometers to 6 significant figures.
- The only perturbations included in the right-hand side are the gravitational ones. This makes it much easier to replicate correctly.

1.5 Model parameters

Since the only perturbations included are gravitational, the only outside data required for the right-hand side are the positions and velocities of Solar System bodies. This leads to two possible ways of simplifying the model, and one way to make the code more efficient:

- The gravitational formula is used with complex relativistic corrections, rather than with the familiar Newtonian model, in [14]. Using simplifications in-between the two may preserve accuracy and reduce computing time.

- In [14], the influence of the Sun, planets and Moon and largest three asteroids are included in the model. The influence of some of the smaller bodies here may be negligible, and should be investigated.
- These bodies are included in the model via NASA-created ephemeride files. To parse these files into usable coordinates for each time point, an API/library is required, whose efficiency has a major effect on total runtime. Therefore, a review of all suitable libraries should be made to ensure maximum efficiency.

1.6 Aims & objectives

To summarise, the aims and objectives of this project are as follows:

Aims

- Compare and contrast the efficiency and accuracy of several numerical methods, primarily Multistep Collocation Methods, in propagating the asteroid Apophis during its 2029 close approach with the earth.
- Comment on the various different possible models & implementations of the right-hand side, their relative performance, and their relative accuracy.

Objectives

- Find the most efficient configurations of Multistep Collocation Methods.
- Contrast these highest-performing Multistep Collocation Methods to high-performance state-of-the-art integrators.
- Find the most efficient gravitational model, planetary model, and ephemeris library for the propagation of Apophis.

2 Background

2.1 Numerical integration

Standard numerical integrators work with first-order equations. To use (1), we first need to convert it to first-order, by combining $\mathbf{r}$ and its first derivative into one variable. The new model is as follows:

$$\begin{aligned} \mathbf{y}'(t) &= \mathbf{f}(t, \mathbf{y}), \\ \mathbf{y}(t_0) &= \mathbf{y_0}, \end{aligned} \tag{2}$$

In (3.5), $\mathbf{y}$ represents the body's *state vector* - a 6-element vector composed of its position and velocity vectors ($\mathbf{r}(t)$ and $\mathbf{r}'(t)$). Its time derivative $\mathbf{f}$ is therefore composed of its velocity vector followed by its acceleration vector, which is produced by the forces acting on the body. In symbolic terms:

$$\mathbf{f} = \frac{d}{dt}\begin{bmatrix} x \\ y \\ z \\ x' \\ y' \\ z' \end{bmatrix} = \begin{bmatrix} x' \\ y' \\ z' \\ x'' \\ y'' \\ z'' \end{bmatrix}.$$

Here, x, y and z are the three components of $\mathbf{r}$ in a given reference frame.

The core principle behind numerically solving first-order ODEs is that we use formulae that, given $\mathbf{y}(t)$, estimates $\mathbf{y}$ a short time step h later, $y(t+h)$. We can recursively use the formulae to, given the initial state vector $\mathbf{y}(t_0)$, generate a series of estimates for $\mathbf{y}$ along the time period of interest, moving along by h (also referred to as the *step size*) with each step.

2.2 Basic numerical integration - Taylor series

For discussion of numerical integrator methods, we simplify $\mathbf{y}$ to be a scalar function y for ease of reading. This is without loss of generality, as the equations can easily be extended to the vector case.

The derivation of most methods is based on the well-known *Taylor expansion* of y. If y is smooth ([17], 1.1), it can be decomposed into an infinite power series in h involving its derivatives, i.e.

$$y(t+h) = y(t) + hy'(t) + \frac{h^2}{2!}y''(t) + \frac{h^3}{3!}y'''(t) + ... + \frac{h^n}{n!}y^{(n)}(t) + ... \quad (3)$$

Given a small h, due to the factorial terms in the series, the terms rapidly decrease. This means that $y(t+h)$ can be very accurately approximated by only using the first few terms of its Taylor expansion, known as *truncating* the series to a given number of terms.

One class of numerical methods, the Taylor-series methods, use this principle directly. Given $y(t_0), y'(t_0), ..., y^n(t_0)$, we approximate $y(t_0+h)$ by the first n terms of its Taylor expansion. The approximation for $y(t_0+h)$ is used in the same way to estimate $y(t_0+2h)$, and so on, until the end of the specified time range is reached.

Using a Taylor-series method with n terms, the error in each step, or *local* error, is guaranteed to decrease at least as fast as h^{n+1} when reducing h. The number of steps is inversely proportional to h, so the total integration error, or *global* error, decreases only at least as fast as h^n. Local and global error are both types of *truncation error*, error caused by the series truncation.

We classify methods with this property as having an *order* of n, denoted as $\mathcal{O}(h^n)$. The order of a method measures how quickly the global error reduces as step size is reduced, which gives one pointer of its accuracy. However, as the magnitude of the truncated terms is unknown, it is not conclusive.

The derivation of Taylor-series methods is very mathematically simple. The catch is that we need to analytically know the first n derivatives of y. Few functions are unable to be integrated analytically, yet easily differentiable analytically n times. Consequently, Taylor-series methods are rarely used in computing - except for computations requiring extremely high precision above all else. They are not studied as part of this project. However, the theory for the methods we will use is built on the Taylor expansion.

2.3 Basic numerical integration - Runge-Kutta and multistep methods

Runge-Kutta methods are an extremely popular family of *multistage* numerical methods, which rely on evaluations of f. At a number of time points between t and $t+h$, an estimate of y is created, which is passed into f to estimate y'. Each y estimate is created using every other y' estimate, as well as $y(t)$. All estimates are then used to approximate $y(t+h)$. For clarity, the general formulation of Runge-Kutta methods is ([19], 2.5.1):

$$\begin{aligned} g_i &= f(t_n + c_i h, y(t_n) + h\sum_{j=1}^{s} a_{ij} g_j)), \text{ for } i = 1..s, \\ y(t_{n+1}) &= y(t_n) + h\sum_{i=1}^{s} b_i g_i, \end{aligned} \tag{4}$$

with the condition that

$$c_i = \sum_{j=0}^{s} a_{ij} \text{ for } i = 1..k.$$

Here, the c_i coefficients represent the time intervals picked to compute f using estimates of y. The a_{ij} coefficient represent the weighting of g_j used to compute g_i, and the b_i coefficients are the final weightings used to estimate y.

The principle here is that by choosing precise a, b, and c coefficients, the Taylor series of the resulting formula is equal to that of $y(t+h)$, at least to the first n terms, to make its local error $\mathcal{O}(n+1)$. An example of how to prove the order of a Runge-Kutta method is given in ([17], 8.3).

The most obvious configuration for the a coefficients is to have $a_{ij} = 0$ for all $j > i$, allowing the g_i to be directly computed consecutively. These methods are called *explicit*.

Runge-Kutta methods where this doesn't hold, i.e. there are two g estimates that rely on each other to be computed, are called *implicit*. The g_i form a set of equations for which root finding techniques must be used. Implicit methods can offer a higher degree of focus on the end of the time interval, which gives high accuracy for stiff equations. The downside is that the root-finding methods during each step incur further f evaluations and computational cost.

A collocation method is a subset of Implicit Runge-Kutta (IRK) methods, and is one which can be visualised more intuitively. Given s time points of

a function, the unique polynomial of $\mathcal{O}(s)$ whose derivative matches, or 'co-locates', with the estimates of f at these points, is integrated instead. By definition, the local error term is then certain to be at least $\mathcal{O}(s+1)$ [6]. By specifically selecting c_i, these methods can reach orders of $2s$.

One important class which will be used is Gauss-Radau or just Radau methods, which have c_k fixed at the end point $t+h$ and all other c_i selected optimally. These can achieve $\mathcal{O}(2s-1)$.

The other major family of numerical methods are *multistep* methods. Instead of approximating the y' values inside the current time interval, multistep methods take already-computed y values at multiple previous time points to move forward to the next. The weightings of these points are chosen exactly to produce a matching Taylor series to a certain order, similarly to Runge-Kutta methods. The general form of a multistep method with r total points is given in ([8], p. 107) and shown below.

$$y_{n+1} = \sum_{i=1}^{r} \alpha_i y_{(n+1)-i} + h \sum_{i=0}^{r} \beta_i f(t_{(n+1)-i}, y_{(n+1)-i}). \tag{5}$$

Multistep methods can also be explicit, or where $\beta_0 \neq 0$, implicit. Implicit multistep methods have been found [7] to be more accurate than the explicit type for a wide variety of inputs.

2.4 Multistep Collocation Methods

Runge-Kutta and multistep methods are both well-studied and state-of-the-art differential equation solvers in numerical computing. Multistep Collocation Methods (MCMs) are a more novel family of methods, which combine the ideas of both Runge-Kutta methods and multistep methods. The basic idea is the same as a Runge-Kutta method of s stages, but further employing the values of r previous solution points during each stage.

The algorithm is an extension of collocation methods - we find a collocation polynomial which matches with k past solution points, and whose derivative matches f at s intermediate solution points; i.e.

$$\begin{aligned} u(t_j) &= y_j & j &= n-k+1,..,n \\ u'(t_n + c_i h) &= f(t_n + c_i h, u(t_n + c_i h)) & i &= 1,..,s, \end{aligned} \tag{6}$$

which gives the numerical solution $y_{n+1} = u_{t_{n+1}}$ [3]. However, as we don't know $u(t_n + c_i h)$, we find estimates of u for each i in the same way as in the classical Runge-Kutta method. The full method is described as such, and can be viewed as a composition of equations 4 and 5:

$$\begin{aligned} g_i &= \sum_{j=1}^{k} a_{ij} y_{n-k+j} + h \sum_{j=1}^{s} b_{ij} f(t_n + c_j h, g_j), \quad i = 1,2,..,s, \\ y &= g_s, \end{aligned} \tag{7}$$

where a_{ij} and b_{ij} are the weighting coefficients.

Similarly to classical Runge-Kutta methods, MCMs are explicit when b_{ij} is strictly lower triangular.

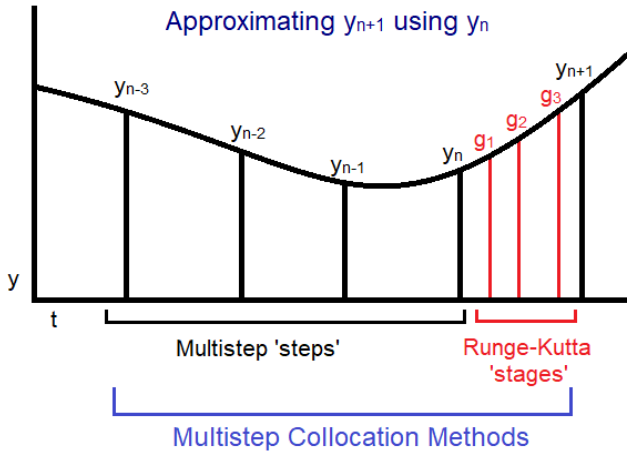

Figure 2: The different families of numerical methods discussed in Section 1.

Figure 2 is a pictorial representation of how multistep and Runge-Kutta methods work, and how MCMs combine them. In this case, we employ four past solution points, and have four stages (g_1, g_2, g_3 and finally, $g_4 = y_{n+1}$).

MCMs are far newer and less studied than Runge-Kutta and multistep methods, which makes them ideal for deep analysis. The idea behind them is the combination of both theories can achieve a very high order, while utilising both the computational cheapness of explicit multistep methods, and the high accuracy at high step size of implicit Runge-Kutta methods. The number of stages (s) and past solution points (k) will dramatically vary the resulting MCM's properties.

This background further clarifies the primary objective of finding the most efficient 'configurations' of MCMs. This means finding the most efficient combinations of k and s to use.

3 Methods

3.1 Multistep Collocation Methods implementation

The MCM implementation used is a Python class called 'MultistepRadau', written by Pete Bartram for [3]. The multistep part of the method is explicit, while the Runge-Kutta part of the method is implicit, using Radau sampling locations c_i. It is shown in [3] that these locations are found, optimising the order, by

solving the following system of equations:

$$\sum_{j=1}^{k} \frac{1}{c_i - t_j} + \sum_{j=1, j\neq i}^{s} \frac{2}{c_i - c_j} = 0, i = 1, 2...s - 1, \tag{8}$$

where $t_1...t_k$ are the previous time steps, and the solution where $0 < c_1 < c_2 < ... < c_{s-1} < 1$ is chosen. The maximal order that these locations obtain for the method is $2s + m - 2$.

The a_{ij} and b_{ij} coefficients are the same for each step - the computation is done at the start, so the cost in efficiency is negligible when there's a large number of steps. In this implementation, they are defined as the solutions given from equations in [26].

The resulting formula is implicit, so we arrive at a system of equations rather than an algorithm, which requires a root-finding algorithm to solve. Newton's method is a simple way to find the root of a single equation, given an initial guess x_0:

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}. \tag{9}$$

Newton's method is the most common for solving ODEs ([23], 17.5), chosen over faster Householder's methods due to its ease in extending to the multivariate case. This is done by extending the $\frac{1}{f'(x_n)}$ term to the inverse of the Jacobian matrix of $\mathbf{f}$, $J^{-1}(\mathbf{f})$ ([17], 3.2). This root-finding system is used to determine the solutions to the series of equations defined by isolating the Runge-Kutta part of the system. We denote the purely Runge-Kutta part of each coefficient as $\mathbf{z}_i$, i.e.

$$\mathbf{z}_i = h \sum_{j=1}^{s} b_{ij} \mathbf{f}(t_n + c_j h, \mathbf{g}_j) = \mathbf{g}_i - \sum_{j=1}^{k} a_{ij} \mathbf{y}_{n-k+j}, i = 1, 2, .., s. \tag{10}$$

This implementation actually uses a simplified version of Newton's method. Here, the Jacobian is only calculated at periodic intervals, rather than during every Newton iteration. The initial guess, or *predictor*, for the root-finding, is extremely important in terms of minimising the number of function evaluations. The implementation uses one of two predictor formulae.

Predictor 1 fits a polynomial through all previous solution points (the multistep part). The polynomial is then evaluated at all spacing time points $c_i h$ to get an initial guess for g_i - the multistep part is then subtracted for a guess for all z_i. Its order is $k - 1$.

Predictor 2 fits a polynomial through all the z_is of the second to last step (the Runge-Kutta part). The difference between y_n and y_{n-1} is added. This polynomial is then evaluated at each new $c_i h$ point. Predictor 2 has order s.

Figure 3 shows when each predictor is employed - Predictor 1 is roughly used for high k and Predictor 2 for high s, to maximise order. In the special case that s=k=1, where there are no previous steps or multiple stages, the z-values from the previous step are used as the guess.

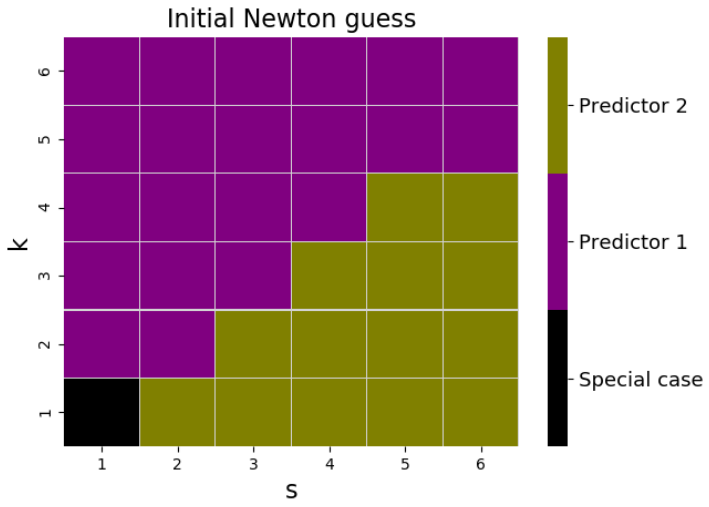

Figure 3: Predictor used for each $k - s$ configuration

3.2 The SciPy integrators

The standard ODE solvers in the SciPy package will be used as a comparison. The new version of this API contains five integrators [27], detailed below, and collected in the function 'solve_ivp'. It was introduced in SciPy 1.0.0, in October 2017 [24].

- The 'BDF' solver is based on the Backward Differentiation Formulae - a class of implicit multistep methods - of orders 1 to 5. The BDF with order s finds the set of s-order polynomials that interpolate the previous $s-1$ points, leaving one degree of freedom in y_n. We then choose y_n such that the polynomial's derivative matches $f(y_n, t_n)$[9]. The error in each step is used to determine which order formula will be used for the next.

- The 'LSODA' solver is a composite integrator, switching between the BDFs (for stiff periods) and the Adams class (for non-stiff periods) of implicit multistep methods, based on detecting stiffness in the equation. Unlike the other fully Python integrators implemented, it's a wrapper for FORTRAN code.

- A common way to implement Runge-Kutta methods is to run two, of orders n and $n+1$, side by side. The difference of the two methods produces an estimate of the nth order term of y, which is used to estimate error. SciPy includes two such implementations with explicit methods. The function 'RK45' uses the Dormand-Prince pair [29], fourth- and fifth- order methods running side-by-side. The function 'RK23' uses the Bogacki-Shampine pair [5], second- and third- order methods running side-by-side.

In both cases, the pair is well-known and chosen specifically for minimising the error of the higher-order method, which is the one that's actually used in the result, while the lower-order method estimates this error.

- The 'Radau' function is part of a family of implicit Runge-Kutta methods with Radau sampling locations. The sampling locations c_i are shown to be the solutions of an algebraic equation ([15], IV.5). This function implements a fifth-order Radau method, by using a third-order collocation polynomial ($s = 3$).

The method for each integrator above is widely used in scientific computing. MATLAB has its own versions of RK45, RK23 and BDF [30]. R includes all the SciPy methods as part of its 'ode' function [22].

3.3 Step size & error estimates

Three types of error, from the 'true' solution, are encountered during numerical integration.

- Truncation error refers to that inherent in the numerical method, caused by the terms in the solution's Taylor expansion which don't match the integrator's. This has been already described thoroughly in Section 2.2.

- Another type of error is caused by the computer itself, called *roundoff* error. In this project, numbers are stored in 64-bit IEEE-754 double precision, which allows 53 bits (approximately 16 decimal digits) available for the significand, and 11 for the exponent [31]. The precision limits incur a very small error in each calculation done, which builds over a large number of calculations ([17], 2.1). Problems are particularly sensitive to round-off error when large numbers are used for calculations, as the larger exponents lower the absolute precision.

- Implicit methods, with root-finding algorithms to solve during each step for coefficients, have an error associated with each set of equations solved. This is particularly apparent with solving systems of vector equations, where solvers attempt to produce the correct solution for each component and for each equation.

Each integrator in Section 3.2 has a way of estimating the truncation error during each step. Normally, this controls step size. The user defines the *tolerance* - a measure of the required bound on minimum error estimate - and before each step, the solver adjusts h based on this and the previous step's error estimate. These are called *adaptive step size* methods. In the 'BDF' and 'LSODA' solvers, which combine multiple methods, the error estimate also determines which method to use for the next step.

Two types of tolerance can be entered into adaptive step size solvers. Absolute tolerance is a constant number to keep the error estimate under, and

relative tolerance gives a factor of the current value of $\mathbf{y}$ to keep error under at each step. Both can be given as scalars, or component-wise vectors.

The issue with relative tolerance is that when $\mathbf{y}$ passes over 0 on any of its components, that component's error bound may become exceedingly strict. For this reason, only absolute tolerance will be used. The components of $\mathbf{y}$ also each have different scales - especially the divide between the first three (position) and last three (velocity) components. For this reason the component-wise form of absolute tolerance will be used.

The method to give give tolerance will be to average the magnitude of each component of the right-hand side, throughout the propagation, collected in a 'characteristic' vector. Absolute tolerance for adaptive step size solvers will then be given in various multiples of this vector.

A problem is that the implementation of Multistep Collocation Methods does not yet have an adaptive step size functionality, whereas the SciPy integrators were designed to primarily be adaptive step-size. Since the primary aim of this project is to compare the underlying methods, step size must be fixed in order for a fair comparison. Therefore, the main comparisons I'll make with the SciPy integrators will be with those programmed for constant step size.

For the explicit 'RK45' and 'RK23' integrators, fixed step size could be implemented properly. These are compared to the MCMs in Section 5.1.

For the 'BDF' and 'LSODA' solvers, step size and method were tied together with the tolerance. This means that fixed step size could not be implemented without artificially reducing efficiency. This means that the underlying *methods* to estimate steps cannot be compared with MCMs. The *integrators* themselves will still be compared with the fixed-step MCM implementation, using the absolute tolerance formula above, in Section 5.5.

The 'Radau' method has the tolerance of its root-finding formula linked to its relative tolerance. This means that the efficiency is artificially reduced by fixing step size, and also that the wholly absolute tolerance method above does not work. For these reasons, the Radau method will not be used in the results.

Now the technicalities of the integrators have been discussed, the actual right-hand side model, and code, can be outlined.

3.4 Gravitational formulas

The right-hand side we will use is solely composed of the n-body gravitational influence from Solar System bodies on Apophis (justified in Section 1.4. The well-known Newtonian formula for the gravitational influence of n bodies on an object of interest is:

$$\ddot{\mathbf{r}} = G \sum_{i=1}^{n} \frac{m_i(\mathbf{r}_i - \mathbf{r})}{r_i^3}. \tag{11}$$

Here,

- m_i represents the mass of body i.

- $\mathbf{r}_i$ represents the distance between body i and the Solar System barycenter.
- $\mathbf{r}$ represents the distance between the object of interest and Solar System barycenter, with magnitude r.
- $r_i = |\mathbf{r}_i - \mathbf{r}|$.
- G represents the Newtonian gravitational constant.

Note that in this case, the problem is greatly simplified as we use downloaded data sets for the positions of the n bodies. We assume that Apophis is sufficiently light to avoid having a significant effect on any of them. This is known as the *restricted* $(n+1)$*-body approximation* [35], where Apophis is body $n+1$.

In reality, due to the high speeds and masses of the bodies, the theory of relativity will noticeably affect the gravity on Apophis. The full relativistic n-body model used in [14] can be written as a power series in $\frac{1}{c^2}$, where c is the speed of light in a vacuum. The zeroth-order terms (i.e. independent of c) are just equivalent to (11), and the Einstein-Infeld-Hoffmann, or EIH, equations, give the zeroth and first order terms of this series [35] - i.e. up to $\mathcal{O}(\frac{1}{c^2})$. The EIH equations are given below:

$$\begin{aligned}\ddot{\mathbf{r}} = G\sum_{i=1}^{n}\frac{m_i(\mathbf{r}_i-\mathbf{r})}{r_i^3}\Big(1 - \frac{2(\beta+\gamma)}{c^2}G\sum_{j=1}^{n}\frac{m_j}{r_j} - \frac{2\beta-1}{c^2}G\sum_{j=1,j\neq i}^{n}\frac{m_j}{r_{ij}} + \frac{\gamma|\dot{\mathbf{r}}|^2}{c^2} \\ + \frac{(1+\gamma)|\dot{\mathbf{r}}_i|^2}{c^2} - \frac{2(1+\gamma)}{c^2}\dot{\mathbf{r}}\cdot\dot{\mathbf{r}}_i - \frac{3}{2c^2}[\frac{(\mathbf{r}-\mathbf{r}_i)\cdot\dot{\mathbf{r}}_i}{r_i}]^2 + \frac{1}{2c^2}(\mathbf{r}_i-\mathbf{r})\cdot\ddot{\mathbf{r}}_i\Big) \\ + G\sum_{i=1}^{n}\frac{m_i}{c^2 r_i}(\frac{3+4\gamma}{2}\ddot{\mathbf{r}}_i + \frac{\{[\mathbf{r}-\mathbf{r}_i]\cdot[(2+2\gamma)\dot{\mathbf{r}} - (1+2\gamma)\dot{\mathbf{r}}_i]\}(\dot{\mathbf{r}}-\dot{\mathbf{r}}_i)}{r_i^2}).\end{aligned} \tag{12}$$

Some extra terms need to be defined here:

- $\mathbf{r}_{ij}$ represents the displacement of body i from body j.
- β and γ are called the *Post-Newtonian Parameters*, which are both equal to one for these equations [28].

Equation (12) is clearly much more complex and time-consuming than the simple Newtonian formula, which may cause problems in propagations long/fine enough, or with a sufficiently complex right-hand side. A less complete, but computationally faster, version of (12) can be found in [4]. The principle here is that we only include relativistic terms proportional to the mass of the Sun - and assume the other planetary masses are sufficiently relatively small to be ignored. Taking m_0 as the mass of the Sun, this leaves us with a rather nicer looking equation for $\ddot{\mathbf{r}}$:

$$\ddot{\mathbf{r}} = G\sum_{i=1}^{n}\frac{m_i(\mathbf{r}_i-\mathbf{r})}{r_i^3} + \frac{Gm_0}{c^2 r^3}\Big((4\frac{Gm_0}{r} - \dot{\mathbf{r}}^2)\mathbf{r} + (\mathbf{r}\cdot\dot{\mathbf{r}})\dot{\mathbf{r}}\Big) \tag{13}$$

A further simplification from [4] takes the target body's eccentricity as zero. This is useful as it follows that $\mathbf{r} \cdot \dot{\mathbf{r}} = 0$, and $\dot{\mathbf{r}} = \sqrt{\frac{Gm_0}{r}}$ ([11], p. 81). This produces the following formula:

$$\ddot{\mathbf{r}} = G\sum_{i=1}^{n} \frac{m_i(\mathbf{r}_i - \mathbf{r})}{r_i^3} + 3(\frac{Gm_0}{c})^2 \frac{\mathbf{r}_i}{r_i^4} \tag{14}$$

This may not be accurate for Apophis, which has eccentricity 0.19 at the initial epoch. In Section 4.2, the propagation is tested using each gravity model.

3.5 Analytic and numerical Jacobians

Implicit methods - Radau, BDF, LSODA and the MCMs for this project - require a Jacobian of the right-hand side for their root-finding method for the implicit series of equations. There are two ways they can get this. The Jacobian can be written as a separate function - called the *analytic* Jacobian. If an analytic Jacobian is not given, it is estimated using a finite difference method, based on evaluating the right-hand side function with small perturbations of each input component in turn. This is called the *numerical* Jacobian.

Ordinarily, an analytic Jacobian is preferable when it is easy to evaluate i.e. the right-hand side is easily differentiable, with respect to each of its components. This is true for the Newtonian case, but progressively less so for the more complex cases. For the Newtonian equations the analytic Jacobian is relatively simple. The Jacobian for the influence of one body, whose gravitational parameter is denoted μ, where Apophis's relative position is $\mathbf{r} = \begin{bmatrix} x \\ y \\ z \end{bmatrix}$, and $r = |\mathbf{r}|$, is:

$$J = \begin{bmatrix} 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 3\mu x^2 r^{-5} - \mu r^{-3} & 3\mu yxr^{-5} & 3\mu zxr^{-5} & 0 & 0 & 0 \\ 3\mu xyr^{-5} & 3\mu y^2 r^{-5} - \mu r^{-3} & 3\mu yzr^{-5} & 0 & 0 & 0 \\ 3\mu xzr^{-5} & 3\mu yzr^{-5} & 3\mu z^2 r^{-5} - \mu r^{-3} & 0 & 0 & 0 \end{bmatrix}. \tag{15}$$

This is created by converting the Newtonian equation to the first-order version in (), rewriting $\mathbf{r}$ in terms of all its components, and taking partial derivatives of each component with respect to each. The full Newtonian Jacobian is created with a summation over all planets for the lower-left quadrant.

3.6 NASA's ephemerides

Now the question comes to how to obtain the planetary coefficients. The most popular ephemerides for the bodies of the Solar System are in the NASA-created

DE (Development Ephemerides) series. The specific file used in [14] is called DE405, released in 1998. The full version of DE405 is an ephemeris of the Sun, planets, Moon and Pluto from 1600 AD to 2200 AD [33]. The outer planets are represented by their barycenters, to include the influence of their numerous moons.

The DE series is built from sets of data points, each containing a body's position and velocity. To accurately interpolate the entire space, segments of between 5 and 15 points, with times normalised between -1 and 1, are each fit to a power series of Chebyshev polynomials. The coefficients for these form the actual data stored in the ephemeris file [20] - this means functions can retrieve a planet's state at any time rather than only at discrete times.

The bodies used in [14], and consequently in this project, are the Sun, planets, Moon, and the 'Big Three' asteroids: Ceres, Pallas and Vesta. The full relativistic gravitational model with these bodies is known in literature as the Standard Dynamical Model [35] [14].

The interface between the integrators and the ephemerides is discussed in Section 4.5. DE405 was used to obtain the positions, and velocities of the planets, Moon and Sun. JPL also has three individual ephemerides for the asteroids, written around the same time as DE405. For their accelerations in the full relativistic model, the difference quotient based on velocities at adjacent times is used. For the gravitational parameters of planets (Gm_i) and c, the same values as used in the creation of DE405 [33] are used.

3.7 Initial conditions

The initial conditions in [14] are given as heliocentric (Sun-centred) Keplerian elements in the ecliptic J2000 frame. They are given in Table 1. (TCB is short for Barycentric Dynamical Time, and is also the time scale used in DE405).

Element	Value	Units
Eccentricity	0.1910573105795565	-
Perihelion radius	0.7460599319224038	AU
Longitude of ascending node	204.45996801109067	Degrees
Argument of perihelion	126.39643948747843	Degrees
Inclination	3.33132242244163	Degrees
Mean anomaly	61.41677858002747	Degrees
Epoch	2453979.5	Julian Days TCB

Table 1: Initial conditions as Keplerian elements

Conversely, DE405 returns a body's position and velocity as Cartesian barycentric (with respect to the Solar System's centre of mass) coordinates, in the J2000 equatorial frame. Cartesian coordinates are also the simplest to numerically integrate with. For optimal efficiency, it's required to convert the initial conditions to this frame prior to propagation. This is done in three stages.

Firstly, to convert between Keplerian elements and Cartesian coordinates, Spiceypy's 'conics' function is used, double-checking with the known formulae in the literature [11]. To convert between astronomical units and kilometers, the au constant used in the creation of DE405 was used.

Secondly, the conversion between the ecliptic and equatorial J2000 frames is a simple rotation, using the earth's obliquity to the ecliptic at J2000 as the angle. The version of this constant from the Astronomical Almanac, 2007 [21] - the year [14] was written - was used.

Finally, to convert between heliocentric and barycentric coordinates, the DE405 Sun state vector at the initial epoch is added to the final result.

Symbol	Meaning	Value	Units
c	Speed of light (vacuum)	299792458	ms^{-2}
au	Astronomical unit	149597870691	m
ϵ_0	J2000 Obliquity to the ecliptic	23.4392911111111	Degrees

Table 2: Constants used in the propagation

3.8 Close approach

Now that the initial conditions $\mathbf{y}_0$, and right-hand side model $\mathbf{f}$, are known, we can propagate to the fly-by region. For the comparison of different integration parameters, and models, we require the closest approach distance to a high precision. This means that the *time* of closest approach must be known very accurately. An iterative method is used to zero-in on the closest approach, as follows:

- Using a constant time step analysis, the solution is known to be between JD 2453979.5 and 2462245.5 (April 8 2029 00:00 and April 18 2029 00:00). The main propagation is made from the initial conditions in 2006 to April 8, with a constant time step h.

- The solution is integrated between April 8 and April 18 using a time step $\frac{h}{100}$. At each point the distance from Earth's DE405 position is computed. The state vector y_1 with minimum distance, at time t_1, is returned. The true time of closest approach is now certain to be in the region $(t_1 - \frac{h}{100}, t_1 + \frac{h}{100})$.

- This is where the recursion starts. The new integration region is taken to be $(t_1 - \frac{h}{100}, t_1 + \frac{h}{100})$, and the time step is divided by 100 again. The closest approach region is again found, to become the new integration region, and is 100 times finer than the last one. This may be repeated arbitrarily. (Multistep methods require sequential computation to be effective. This is why this recursion method was chosen over the more simple binary search.)

- The relative velocity of Apophis, with respect to the Earth, is computed using y and DE405. Multiplying this by half of the final time step gives an upper bound on the error remaining in y. The progressive refinement may be increased until the desired precision is obtained - this code uses four iterations.

For example, with a step size of 1 day = 86400 seconds, the closest approach with four iterations is found to an accuracy of 0.000864 seconds. Figure 4 shows the error in close approach caused by a time error of a few seconds is in the 10m range, and that therefore the time accuracy we have chosen is more than acceptable to remove discretisation error.

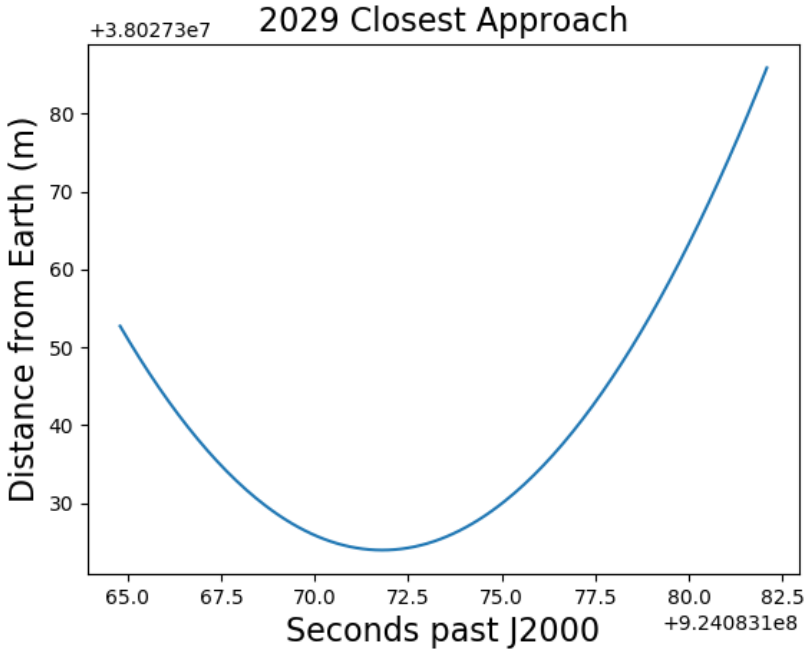

Figure 4: Distance of Apophis from Earth due to the 2029 close approach

4 Results & discussion: The right-hand side

4.1 Validating the right-hand side

The 'correct', most precise, distance of closest approach found was 38027.525 km, to the nearest metre. This was computed by the 'RK45' integrator, using the relativistic equations of motion (12), and the Spiceypy library. The precision was verified by halving the step size until the difference

between consecutive propagations was within 1mm - the final step size was 64 steps per day. I'll refer to this as A_0 from here on.

The RK45 integrator was chosen to calculate A_0 for the following reasons:

- It is a state-of-the art integrator with mature code, unlike the MCM implementation.
- It uses an explicit method, which rules out any error from a root-finding system for implicit methods - although the implicit methods may be faster.
- It is higher-order than RK23, the other explicit-method integrator - and as a result, is later shown to be far faster in Section 5.1.

The closest approach found in ([14], Table 5) was given to 3 significant figures as 0.000254 AU, i.e. in the region (0.0002535, 0.0002545) AU. This region is equivalent to (37920, 38070) km, which this project's closest approach was within. However, the closest approach *time* was off by 3 seconds.

The initial conditions from [14] have also been propagated in [18] to the time they gave, giving a closest approach distance of 38027.2km (6.s.f). This is a 300m discrepancy from A_0, which can't be explained by the 3-second time difference - as Figure 4 shows.

Implicit equation solving error has been eliminated by choosing RK45, and used the precise values of each constant in the model and initial conditions conversion (the mass values and Table 3.7). However, these discrepancies may still be caused by differing values of these constants with the two papers. The other possible reason is that significant round-off error may accumulate during the propagation, with the very large number of calculations and the Solar System-level distances being worked with.

Given the verification of the right-hand side, we now have a great differential equation set up to test the integrators with. Firstly, the secondary aims of testing model parameters can be completed - this will make the integrator testing, the bulk of this project, substantially more efficient.

4.2 Which gravitational model is most suitable?

For the rest of the work done in this project, it is very useful to know which gravity model from Section 3.4 is least time-consuming, while remaining accurate enough to be useful in comparing integrators. Each model was first run at very low step size to determine the closest approach they converge to, then compared to A_0 (the full relativistic solution). Then, they were each run ten times with 8 steps per day, with the RK45 integrator, to determine their mean propagation time.

Table 3 shows that the first light relativistic model clearly comes much closer to the full one than the other two models, while Table 4 shows it retains a huge decrease in propagation time. We conclude that using the first light relativistic model, accurate to within 200 metres, is best for comparing integrators. For the rest of the report I'll be using this model. The convergent closest approach from it will be denoted A_1.

Gravitational model	Distance from A_0 (m)
Full relativistic (12)	0
Light relativistic (13)	-176
Light relativistic (e=0) (14)	+15350
Newtonian (11)	-1015716

Table 3: Closest approaches using various gravitational models

Gravitational model	Mean propagation time	Standard deviation
Full relativistic	774.622	10.109
Light relativistic	194.578	3.508
Light relativistic (e = 0)	189.133	1.705
Newtonian	171.853	1.887

Table 4: Performance of various gravitational models

4.3 Using different ephemerides

It is also interesting to see the effect that the accuracy of the ephemeris has on the approach. This gives a rough estimate on the error in closest approach from the imperfections in the computed ephemerides. Two newer NASA ephemerides were tested:

- DE421 was a major update from DE405 by JPL, and one which added new 'best estimates' of planetary orbits and masses.
- DE430 replaced DE405 for use in the Astronomical Almanac from 2014.

Ephemeris	Divergence from A_1 (m)
DE430	+685
DE421	-455
DE405	0

Table 5: Effect of different ephemerides on closest approach

Table 5 shows that changing the planetary ephemeris does not have a major effect on the closest approach (<1 km).

4.4 Which Solar System bodies most affect the propagation?

It is useful to see which Solar System bodies the propagation is most sensitive to. This not only verifies the optimal number of bodies for an accurate, yet

efficient, propagation, but gives an idea of the sensitivity of the model to an error in planetary mass and from omitted bodies.

To test this, the propagation with 32 steps per day and the light relativistic model was run, omitting the influence of each planet and the Moon in turn. The distance from A_1 by omitting each body is given in Table 6. When this is larger than 10^8 m, i. e. there is no real close approach in the expected period, N/A is given. To further test the bodies which exhibit this behaviour, the model is retested increasing each body's mass by 0.0001%, shown in Table 7. This shows the influence that the accuracy of the mass values has, especially important since they are being refined constantly as new measurements of planets are taken.

We see that all the major Solar System bodies have a huge effect on the closest approach. Unsurprisingly, the body with the largest influence is the Sun. The largest perturbers are Venus - which has the closest fly-by in the intervening period, Earth - which has two fly-bys during this time, and Jupiter - the largest Solar System planet.

Omitted body	Distance from A_1 (m)
Sun	N/A
Mercury	-192043.9
Venus	N/A
Earth	N/A
Moon	+30123066.0
Mars	-12714459.8
Jupiter	N/A
Saturn	-2344299.8
Uranus	-2295356.8
Neptune	-683125.2
Pluto	-0.4
Ceres	-66.9
Pallas	-53.1
Vesta	-875.7

Table 6: Effect of omitting body from n-body model

Body	Distance from A_1 (m)
Sun	-3056063.3
Venus	+517.0
Earth	-240.9
Jupiter	+136.3

Table 7: Effect of adding 0.0001% onto body's mass

Table 6 further shows the redundancy of Pluto in the model, explained by its vast distance from Apophis and low mass. Ceres and Pallas also do not make much of a difference. For the rest of the tests in this report, Pluto is not included.

Unsurprisingly, the Sun has the largest effect on Apophis by far. Two interesting results are the relatively large effect that Vesta has with respect to the other two asteroids, and that the larger effect that Venus has than Earth (shown in Table 7). This is despite the mass of Venus being less than Earth, and seemingly acting the same amount of acceleration on Apophis in the intervening period. This phenomenon with Vesta was observed to a lesser extent in ([14], Table 7), where Vesta had an unusually large effect compared to Pallas on the 2036 close approach. Figure 5(b) shows the acceleration on Apophis acted on by each of Earth, Venus, Jupiter, and the Sun. This plot shows that the Venus close approach is the most perturbing - verified with ([14], Table 3) which shows it's easily the closest in the propagation region. It is hypothesised that this is the source of the larger effect that Venus has.

Figure 5(a) shows the acceleration by each asteroid, with the acceleration from the Sun normalised and overlaid. It shows that the peaks of Vesta's influence in the propagation period roughly align with Apophis's aphelion. This creates a cumulative resonance effect which magnifies Vesta's influence. Also, perturbations at aphelion have a significantly larger effect than at perihelion. This seems to account for Vesta's unexpectedly large effect,

4.5 Which library is best to interface with DE405?

An initial concern is the way that DE405 is translated into Python to obtain a body's state vector at each required time. Two Python libraries were found which were able to parse DE405 (and other DEs) accurately.

- The Spiceypy library is a wrapper for the C version of SPICE, the JPL produced toolkit for orbital mechanics. SPICE is NASA's recommended way of processing DE files.
- Astropy is a popular, versatile package for general astronomy in python, and includes simple functions [13] for calculating the barycentric position and velocity of bodies, in J2000 equatorial coordinates.

Using both libraries, the light relativistic model (13) was run five times with a step every 2 days to find the mean propagation time.

Library	Mean propagation time	Standard deviation
Astropy	107.4072	0.8746
Spiceypy	10.0063	0.6946

Table 8: Propagation time using the light relativistic gravity model

The difference between the convergent closest approach of each library was under 1 metre. The Spiceypy library was chosen for two reasons:

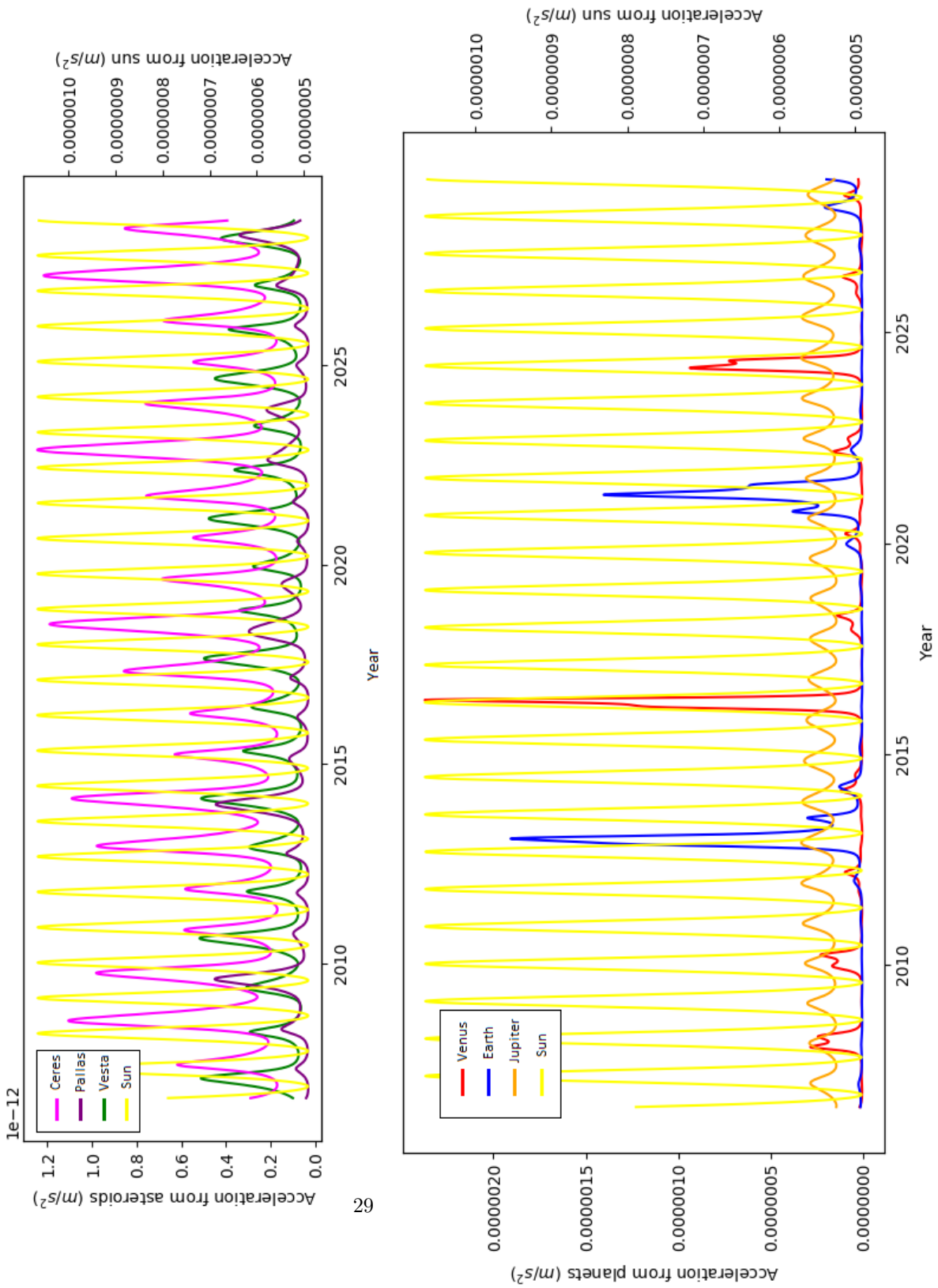

Figure 5: Perturbing acceleration from (a) each asteroid in the model, and (b) the most influential planets, with sun acceleration overlaid

- Table 8 shows that propagations running with Spiceypy ran over ten times faster than equivalent Astropy propagations. This can be explained by Spiceypy being a wrapper for code in C, which is a much faster, low-level programming language than Python. Astropy's object-oriented structure provides another layer of abstraction, which further decreases computing efficiency.

- We also require the loading of multiple ephemerides at once - the DE405 ephemeris for the Sun, Moon and planets, and three individual ephemerides for the Big Three asteroids. This can be done smoothly in Spiceypy, where each ephemeris is loaded at the start into one big data set. In Astropy, the ephemeris must manually be switched during every right-hand side evaluation to the asteroids and back.

The secondary aims of discussing model efficiency and accuracy have been thoroughly covered. Now it is finally time to run tests to compare each numerical integrator.

5 Results & discussion: Comparing the integrators

All time-sensitive results in this report were computed on the same day (for each separate result set), with a 3.6 GHz Intel Core i7 Windows 10 machine with 16GB RAM. Only the integrator code and command line were open.

We denote a MCM with k previous steps and s stages as 'MCMks'.

5.1 How accurate can you get in 20 seconds?

The most intuitive and direct metric for measuring integrator efficiency is time - fixing the time taken, and measuring the accuracy achieved with each configuration. As it's impossible to directly fix time, this was achieved by iteratively adjusting h using an interval bisection method, repeating this until the propagation time was one standard deviation within 20 seconds. This process was repeated three times, with different starting limits for the bisection each time. This was done for MCMs with $1 \leq k, s \leq 6$, as well as for RK45 and RK23. The closest approach was capped at 10^8 metres; configurations which were less accurate are shown in white.

The traditional fixed-step integrators are easily outperformed by the highest order MCMs. The graph mostly follows a trend of being more accurate with higher k and even more so with higher s. This is to be expected as the order is $2s + k - 2$. Two interesting results are the high performance of MCM16, and the anomalous low performance of MCM63.

The number of Newton iterations is an extremely important factor in the efficiency of a configuration, and may provide insight into these results. Measuring the mean number of iterations is an excellent way of isolating the performance

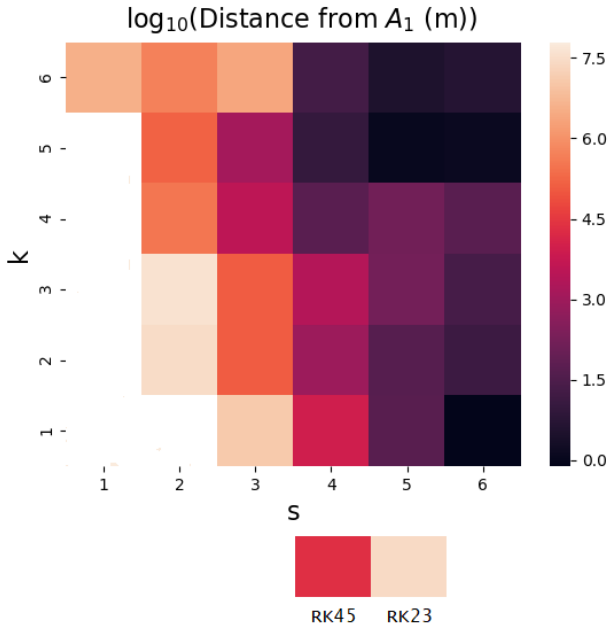

Figure 6: Distance from A_1 with 20 seconds of propagation time

of the predictor. Figure 7 shows the mean number of Newton iterations from the '20 seconds' test above, with the configurations that converged. Configurations which did not converge are shown in black.

The results in Figure 7 shed further light on the trends. Predictor 2 is clearly outperformed by Predictor 1, with the exception of purely Radau configurations. Taking this into account, configurations close to the pure multistep or pure Radau form converge the quickest.

This phenomenon was explained further in [3]. Here it was shown that a big issue in the predictor is the discrepancy in its order, compared to the order of the resulting method, or *corrector*. The two predictors are $\mathcal{O}(k+1)$ and $\mathcal{O}(s)$, compared to the corrector of $\mathcal{O}(k+2s-2)$. The results of the Apophis propagation further confirm this, and its effect on the time efficiency of MCMs.

5.2 Changing the predictor

Results were recomputed using Predictor 1 for $k > 1$, results of which are shown in Figure 8.

We see an immediate improvement in most of the relevant results. However, there are two more clear anomalies in MCM26 and MCM46, as well as the anomaly MCM63 remaining.

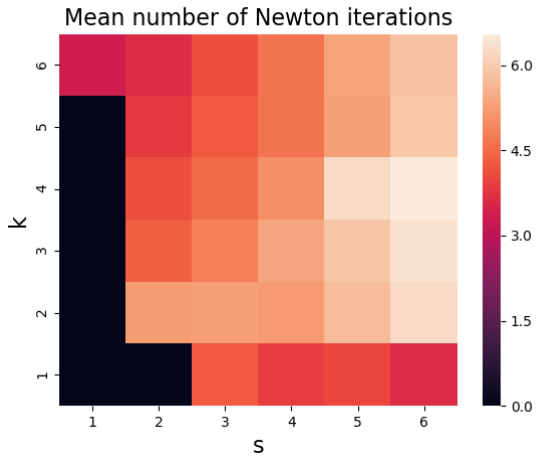

Figure 7: Mean number of Newton iterations.

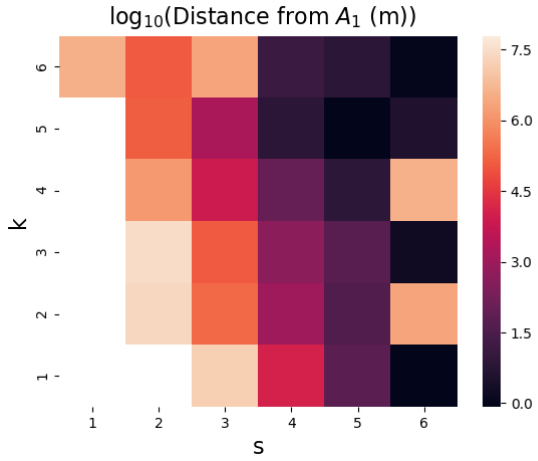

Figure 8: The same '20 seconds' test, using predictor 1 for $k > 1$

5.3 The diverging configurations

As an example, the MCM46 was tested with many different step sizes. Figure 9 shows how the distance from A_1 exponentially decreases when the step size is refined. The general trend is to be expected, but what's really intriguing is the region of step-sizes where most (but not all) propagations diverge about 10^5 further away from A_1 as would be expected.

The large relative error in selected propagations turns out to root from specific steps with an unusually low number of Newton iterations. To patch this,

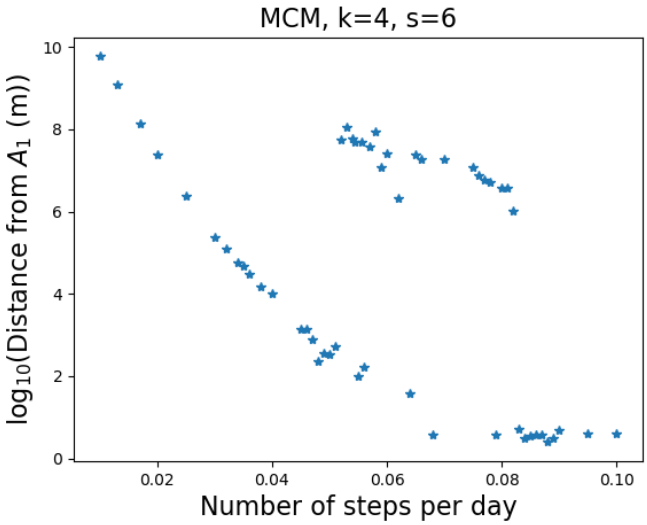

Figure 9: The unusually divergent MCM propagations with $k = 4, s = 6$

an argument to the integrator has been added which set the minimum number of Newton iterations per step. I set this to 4 for the first running of the MCM (the one through the majority of the time period).

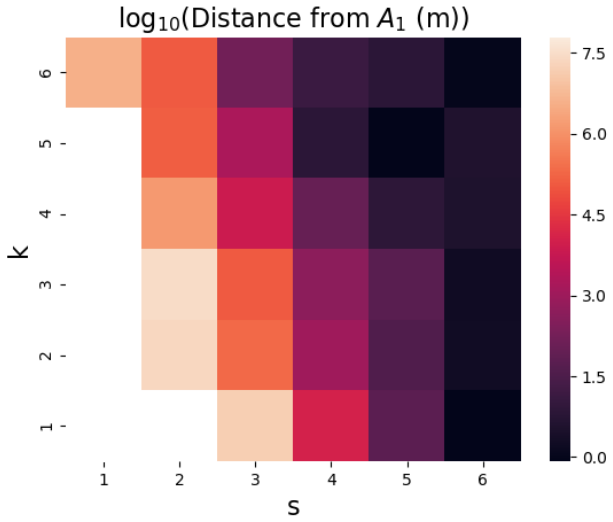

Figure 10: Patching the code by introducing minimum Newton iterations

Figure 10 shows correction of the three divergent configurations. The results

shown in Figure 6 still hold here: propagations are most efficient with a high s, and either a high k or $k = 1$.

5.4 Using an analytic Jacobian

The source of the three anomalies, and the error in their Newton iterations, turns out to root from the way the numerical Jacobian is calculated. The cells of the lower-left quadrant - derivatives of acceleration components with respect to position components - have magnitudes in the range of 10^{-15}, and are susceptible to round-off error when numerically calculated. To solve this, an analytic Jacobian was used with MCM integrators. This is a better fix than the minimum Newton iterations patch, as it is not obvious for future problems what to set a minimum iteration limit to.

Writing an analytic Jacobian becomes extremely complex for the relativistically corrected models, with negligible gain in accuracy. The Jacobian of the light relativistic equations is instead estimated with the analytic Jacobian of the *fully Newtonian* equations.

Integrator	Numerical Jacobian	Analytic Jacobian
MCM55	28.551 ± 0.128	24.021 ± 0.316
MCM16	25.810 ± 0.160	20.185 ± 0.140

Table 9: Mean propagation times of MCM integrators with the two different types of Jacobian input

Integrator	Numerical Jacobian	Analytic Jacobian
MCM16	1.832	1.835
MCM55	2.255	2.137
MCM66	2.015	2.011

Table 10: Average Newton iterations of MCM integrators with the two different types of Jacobian input

Table 9 displays the result of time tests on two MCMs. There's a noticeably larger improvement in the MCMs with an analytic Jacobian. The difference in closest approach between the two types for all three integrators was <30cm. Table 10 - the average number of Newton iterations for selected propagations - shows that an analytic Jacobian does not significantly decrease this number. Therefore, the decrease in time is due to the faster computation of the Jacobian.

As the Newtonian analytic Jacobian is accurate, faster, and not error-prone, it is therefore preferable to use over the numerical approximation to the light relativistic Jacobian.

5.5 How do adaptive step-size methods compare?

In the 'LSODA' and 'BDF' integrators, the order of the method is tightly linked to the step size. Therefore, they are unable to run with a fixed step size. This made them unsuitable and unfair for comparison in Section 5.1, but still interesting to see how they compare. The accuracy versus propagation time, and versus right-hand side evaluations, was compared. This is also a great opportunity to more thoroughly test the three best-performing MCMs: MCM16, MCM55 and MCM66.

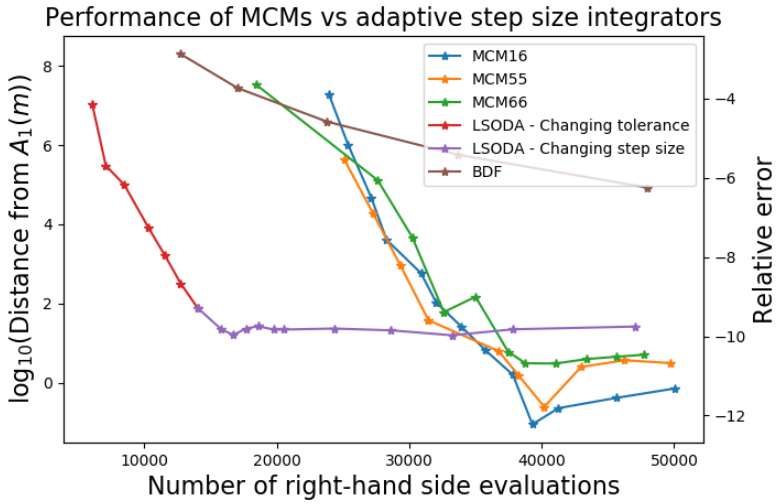

Figure 11: Right-hand side evaluations versus closest approach for selected integrators

Surprisingly in Figure 11, all three MCMs easily outperformed the BDF integrator, even given its handicap of fixed step size. LSODA, which combined the BDFs with the Adams method for non-stiff regions, converged much quicker. However, changing the tolerance alone did not converge LSODA to its maximum level. The points given by changing the tolerance were plotted in red, and when changing the tolerance no longer had an effect, a maximum step size parameter was also added, plotting the points given by refining that in purple.

LSODA was dominant over the MCMs, although did not converge to its end value based on changing tolerance alone. This is a key downside of adaptive step-size integrators - high control of the integration is traded off for greater efficiency. MCM16 and MCM55 performed at similar efficiencies, both outperforming MCM66.

The right-hand axis of Figure 11 shows the *relative error* - the distance from A_1 divided by the magnitude of the actual value being computed - distance from the Solar System barycenter. This shows that the integrators which have converged agree to about the 10th digit.

Round-off error may be reduced significantly by representing the numbers with quadruple precision (128 bits) rather than double precision (64 bits). However, this will not help greatly in producing a more accurate prediction - round-off error is far smaller than the uncertainty in body positions between planetary ephemerides from Section 4.3, and error caused by the omitted perturbations discussed in Section 6.4.

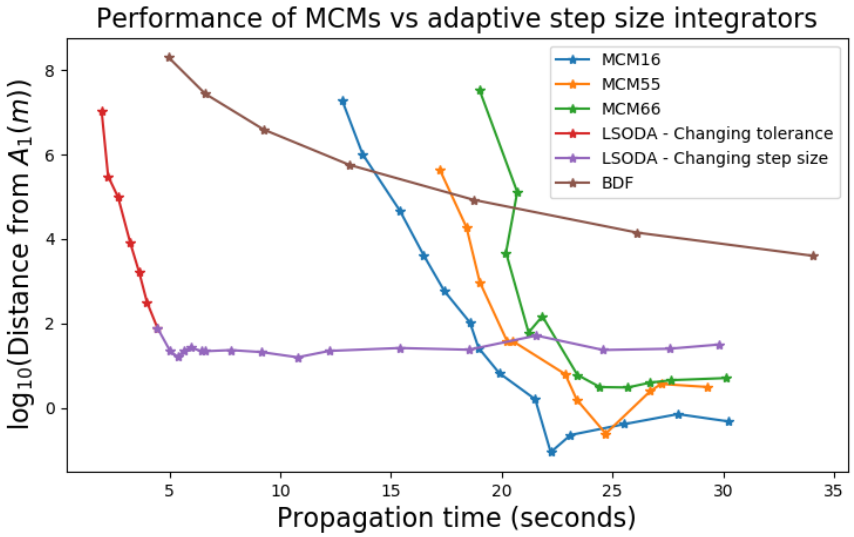

Figure 12: Propagation time versus closest approach for selected integrators

Figure 12 considers propagation time instead of evaluations. LSODA becomes even more dominant here, as a result of the code being a FORTRAN wrapper. Also, MCM16 becomes clearly superior to MCM55. This edge in time, with respect to function evaluations, may be because of the lower complexity of the algorithms of the lower-order, purely Radau, method.

6 Conclusions & further analysis

6.1 Conclusions: The integrators

Properties of MCMs have been investigated in recent literature [3] [36] as a proposed alternative to the current state-of-the-art. In [3] it was concluded that MCMs are most efficient with either a low k & high s, or a low s & high k, and are less efficient with both high k and s, or low k and s.

Another conclusion there was that the parameter space of MCMs make them versatile for use in a variety of problems; i. e. that a $k - s$ combination can be specifically selected for a given problem. The recommendation from this report - fulfilling the primary aims and objectives - is, for an orbital mechanics problem,

to use MCM16 - $k = 1$ and $s = 6$ (i.e. purely Radau) - for this implementation of Multistep Collocation Methods. A high s value seems most important to an efficient integrator, however we have seen that high k, high s methods can be comparable - MCM16 was the only method comparable to the high-order $k, s \geq 5$. The loss in efficiency with higher Newton iterations, due to the order difference between predictor and corrector, has been shown to be counteracted by the much higher order of the method itself. A large k also means that the multistep predictor - shown to be the stronger predictor in Section 5.1 - can be used.

The best-performing MCMs easily outdid the built-in fixed step size Python integrators. Another research topic, left beyond the scope of this project, is to investigate even higher values of s and k for use. It remains to be seen, especially with a more accurate predictor, how high the two variables can be increased without loss of efficiency.

6.2 Potential improvements and further analysis

A key next step is the addition of truncation tolerance and adaptive step-size. This was one of the keys to the only method from SciPy that was able to outperform MCMs.

The wish for a more accurate predictor for the Newton method, which uses both previous steps and previous stages, has already been outlined in [3]. It is likely that creating the recommended predictor of $\mathcal{O}(s + k)$ will push the higher-order MCMs to outdo the purely Radau.

Another clear path for the implementation of MCMs is to implement it in a low-level language. Python is a great language for creating 'proof of concept' codes, and comparing different integrators, due to its simplicity. The trade-off that it's not the fastest language, being an interpretive language rather than compiled. Efficiency could be greatly increased by choosing another language such as C or FORTRAN. Indeed, the very fastest way to propagate Apophis in this project was found by using Spiceypy - a wrapper for a C library - and LSODA - a wrapper for FORTRAN code. One option for MCMs is to implement them in C and create a wrapper in Python. This would retain the ease of use while still gaining a very large in time.

6.3 Conclusions: The right-hand side

The secondary aims and objectives have been fulfilled with the following conclusions:

- For a highly accurate propagation retaining low time cost, the 'light relativistic' model outlined in Section 3.4 is the recommended model to use.
- However, the analytic Jacobian from the Newtonian equations can be used alongside implicit methods, without loss of accuracy. Using this instead of a numerical Jacobian also prevents round-off errors in its calculation.

- Even for very highly accurate propagations, Pluto can be omitted in the gravitational model. We have also seen that bodies can have amplified effects on a trajectory, based on resonance between a body's orbit and the perturbing body's orbit.
- In Python, the Spiceypy library is much better suited for propagation of planets and bodies than the Astropy library.
- The choice of planetary ephemeris does not make an outstanding difference in trajectories, compared to other perturbations.

6.4 Omitted perturbations

The simplified model used in this report includes only the relativistic gravitational influence of the Sun, planets and three asteroids. For high-accuracy predictions, it may be necessary to include several other perturbations:

- There are hundreds more asteroids that have a small perturbing effect, shown in ([14], Table 6).
- The collective force from photons from the Sun hitting a body is called *solar radiation pressure*. Over a period of years, this perturbs Apophis enough to cause a noticable effect in its ephemeris and closest approach [37].
- A further perturbation in rotating bodies is when photons are absorbed and re-emitted a short time later in a different direction, causing an overall change in momentum. This is known as the Yarkovsky effect, and is the largest factor in the uncertainty of Apophis's ephemeris, due to the rotation of its spin being currently unknown [2].
- One of Newton's earliest uses for his theory of calculus was proving the shell theorem - a spherical object, with uniform density, can be treated as a point mass with regard to its influence on an external body. This theorem is the foundation of all orbital mechanics - as it allows the use of the formulae detailed in section 3.4, by modelling the planets as spheres. However, planets are never exactly uniformly spherical - and in particular, the nonsphericity of the Earth and Sun perturbs Apophis's orbit. A power-series approximation to these perturbations for Earth is discussed in ([14], Table 8).

As this project was focused on comparing integrators, rather than creating the most accurate ephemeris possible, these perturbations were not included. This has limited the secondary aim of optimising model implementation, to only comparing choices associated with the gravitational model.

However, the presence of many additional perturbations greatly increases the total propagation time. In this project the propagations converged very quickly, and so choosing a fast and efficient integrator didn't seem to be particularly

important. This changes when a more accurate model is required, with each perturbation listed above added. Here, as conserving time becomes paramount, the selection of a fast integrator becomes exceptionally important.

References

[1] *2004 MN4 Impact Risk*. URL: https://web.archive.org/web/20050314032111/https://astro.uni-bonn.de/~dfischer/mirror/285neo041227.html. (accessed 25.05.2019).

[2] Vokrouhlický D. et al. "The Yarkovsky effect for 99942 Apophis". In: *Icarus* 252 (2015), pp. 277–283. DOI: 10.1016/j.icarus.2015.01.011.

[3] P. Bartram, H. Urrutxua, and A. Wittig. "Exploring the Parameter Space of Multistep Collocation Methods".

[4] G. Beutler. *Methods of Celestial Mechanics, Volume 1: Physical, Mathematical, and Numerical Principles*. Springer, 2005. ISBN: 3540407499.

[5] P Bogacki and F Shampine. "A 3(2) Pair of Runge-Kutta Formulas". In: *Applied Mathematics Letters* 2.4 (1989), pp. 321–325.

[6] H. Brunner. *Collocation methods for Volterra Integral and Related Functional Equations*. Cambridge University Press, 2004. ISBN: 9780521806152.

[7] J. C. Butcher. "A history of Runge-Kutta methods". In: *Applied Numerical Mathematics* 20.3 (1996), pp. 247–260.

[8] J. C. Butcher. *Numerical Methods for Ordinary Differential Equations*. 2nd ed. John Wiley & Sons, Ltd, 2008. ISBN: 978-0-470-72335-7.

[9] G. D. Byrne and A. C. Hindmarsh. "A Polyalgorithm for the Numerical Solution of Ordinary Differential Equations". In: *ACM Transactions on Mathematical Software* 1 (1 1975), pp. 71–96.

[10] S. R. Chesley. "Potential impact detection for Near-Earth asteroids: the case of 99942 Apophis (2004 MN4)". In: Asteroids, comets, meteors : proceedings of the 229th Symposium of the International Astronomical Union. (Búzios, Rio de Janeiro, Brasil). 2005.

[11] H. D. Curtis. *Orbital Mechanics for Engineering Students*. 3rd ed. Butterworth-Heinemann, 2014. ISBN: 9780080977478.

[12] B. Dachwald and R. Kahle. "Head-On Impact Deflection of NEAs: A Case Study for 99942 Apophis". In: Planetary Defense Conference. (Washington, D. C., USA). 2007.

[13] The Astropy Developers. *Solar System Ephemerides - Astropy v3.2.1*. 2019. URL: http://docs.astropy.org/en/stable/coordinates/solarsystem.html.

[14] J. D. Giorgini et al. "Predicting the Earth encounters of (99942) Apophis". In: *Icarus* 193.1 (2008), pp. 1–19. DOI: 10.1016/j.icarus.2007.09.012.

[15] E. Hairer and G. Wanner. *Solving Ordinary Differential Equations II*. Vol. 14. Springer, 1991.

[16] *JPL Small-Body Database Browser*. URL: https://ssd.jpl.nasa.gov/sbdb.cgi?sstr=99942;cad=1. (accessed 05.04.2019).

[17] D. Kincaid and W. Cheney. *Numerical Analysis: Mathematics of Scientific Computing*. 3rd ed. American Mathematical Society, 2002.

[18] O. M. Kochetova, Y. A. Chernetenko, and V. A. Shor. "How precise is the orbit of (99942) Apophis and how probable is its collision with the Earth in 2036-2037". In: *Solar System Research* 43 (4), pp. 324–333.

[19] L. Lapidus and J. Seinfeld. *Numerical Solution of Ordinary Differential Equations*. Academic Press, 1971.

[20] X. X. Newhall. "Numerical Representation of Planetary Ephemerides". In: *Celestial Mechanics* 45 (1989), pp. 305–310.

[21] The Astronomical Almanac Online. *Selected Astronomical Constants*. 2007. URL: http://asa.usno.navy.mil/static/files/2007/Astronomical_Constants_2007.pdf. (accessed 05.08.2019).

[22] T. Petzoldt. *ode function - R Documentation*. URL: https://www.rdocumentation.org/packages/deSolve/versions/1.24/topics/ode. (accessed 15.08.2019).

[23] W. Press et al. *Numerical Recipes: The Art of Scientific Computing*. 3rd ed. Cambridge University Press, 1988.

[24] *Releases - scipy/scipy*. URL: https://github.com/scipy/scipy/releases?after=v1.0.1. (accessed 28.08.2019).

[25] C. M. Rumpf, H. G. Lewis, and P. M. Atkinson. "Asteroid impact effects and their immediate hazards for human populations". In: *Geophysical Research Letters* 44 (2017), pp. 3433–3440. DOI: 10.1002/2017GL073191.

[26] S. Schneider. "Numerical Experiments with a Multistep Radau Method". In: *BIT Numerical Mathematics* 33 (1993), pp. 332–350.

[27] *scipy.integrate.solve_ivp – SciPy v1.3.0 Reference Guide*. URL: https://docs.scipy.org/doc/scipy/reference/generated/scipy.integrate.solve%5C_ivp.html. (accessed 21.07.2019).

[28] P. K. Seidelmann. *Explanatory Supplement to the Astronomical Almanac*. University Science Books, 1992.

[29] F Shampine. "Some Practical Runge-Kutta Formulas". In: *Mathematics of Computation* 46.173 (1993), pp. 135–150.

[30] L. F. Shampine and M. W. Reichelt. URL: https://www.mathworks.com/help/pdf_doc/otherdocs/ode_suite.pdf. (accessed 14.08.2019).

[31] IEEE Computer Society. *754-2008 - IEEE Standard for Floating-Point Arithmetic*. 2008.

[32] L. L. Sokolov. "On the Keyhole Positions of Apophis". In: IAA Planetary Defense Conference. (ESA-ESRIN, Frascati, Italy). 2015.

[33] E. M. Standish. *JPL Planetary and Lunar Ephemerides, DE405/LE405*. 1998.

[34] M. Trenti and P. Hut. "N-body simulations (gravitational)". In: *Scholarpaedia* (2008). DOI: 10.4249/scholarpedia.3930. URL: http://www.scholarpedia.org/article/N-body_simulations.

[35] A. Wittig et al. "Propagation of large uncertainty sets in orbital dynamics by automatic domain splitting". In: *Celestial Mechanics and Dynamical Astronomy* 122 (3 2015), pp. 239–261. DOI: 10.1007/s10569-015-9618-3.

[36] C. Zhang and S. Vandewalle. "General Linear Methods for Volterra Integro-Differential Equations with Memory". In: *SIAM Journal on Scientific Computing* 27.6 (2006), pp. 2010–2031.

[37] J. Žižka and D. Vokrouhlický. "Solar radiation pressure on (99942) Apophis". In: *Icarus* 211 (1 2011), pp. 511–518.