Mars Sample Return ascent trajectory optimisation using Taylor Series integration

Master Thesis

S.D. Petrovic

Faculty Aerospace, Section Spaceflight

TU Delft Delft University of Technology

Challenge the future

PREFACE

Back in December 2013 Warren Gebbett gave a presentation on his work at the Jet Propulsion Laboratory (JPL) in Pasadena, California, USA and the opportunity for a new student to go and perform research at JPL. At this point I sent in my application together with eight other students. Then at the end of December I heard that I was invited for an interview in the first week of January 2014. In this interview it was concluded that I met all the requirements and that I was the perfect candidate to follow Warren up as the next student at JPL with financial backing of Dutch Space (now Airbus Defence and Space, the Netherlands). Financial backing was also going to be provided by the Stichting Prof.dr.ir. H.J. van der Maas Fonds (Aerospace Engineering Faculty, TU Delft) and the Stichting Universiteitsfonds Delft (TU Delft). Communication with JPL was thus started and in March 2015 it was clear that I would be working for the Mars Program Formulation Office under the supervision of Roby Wilson (Inner Solar System group, NASA JPL). He told me to focus on subjects that dealt with Mars missions. At that point I was doing my internship at DLR Bremen on Lunar rocket ascent and descent, which lasted till June 2015. When I came back to Delft me and my supervisors Erwin Mooij (rockets, trajectories, entry and descent, TU Delft) and Ron Noomen (mission design and orbit analysis, TU Delft) agreed that it would be best to perform a study on these Mars subjects to prepare for my visit to JPL and to formulate proposal thesis topics. The first week at JPL I presented these initial thesis topics to both people from the Inner Solar System group and the Mars program formulation office. The next few weeks were spent choosing and refining one of these topics. This document is the result of the two-month literature study on that topic to prepare for the thesis project.

S.D. Petrovic
Pasadena, California, February 2016

CONTENTS

ABBREVIATIONS

ACT	Advanced Concepts Team
DE	Differential Evolution
ESA	European Space Agency
GLOM	Gross Lift-Off Mass
GRAM	Global Reference Atmospheric Model
JPL	Jet Propulsion Laboratory
MAV	Mars Ascent Vehicle
MBH	Monotonic Basin Hopping
MOLA	Mars Orbiter Laser Altimeter
MSR	Mars Sample Return
PaGMO	Parallel Global Multi-objective Optimizer
RF	Frame of Reference
RKF45	Runge-Kutta-Fehlberg 4^{th} (5^{th}) order
RKF	Runge-Kutta-Fehlberg
RK4	Runge-Kutta 4^{th} order
s/c	Spacecraft
SNOPT	Sparse Nonlinear Optimizer
SQP	Sequential Quadratic Programming
TSI	Taylor Series integration
Tudat	TU Delft Astrodynamics Toolbox

1

INTRODUCTION

Mars Sample Return (MSR) has been a mission concept that has been proposed many times in the past two decades. Even today, research into this mission is still being done. And although it is not yet an official project proposal, NASAs Jet Propulsion Laboratory (JPL) is currently working on pre-cursor missions to eventually launch an MSR mission. To prepare for this, research is being conducted on different aspects of MSR, such as the Mars Ascent Vehicle (MAV) responsible for transporting the dirt and soil samples into a Martian orbit and the orbiter which will then transport the samples back to Earth. The current orbiter proposed by JPL is a low-thrust orbiter called Mars 2022. Such an MSR mission requires precise and optimum (optimised for lowest Gross Lift-Off Mass (GLOM)) trajectories to be able to bring back as many samples as possible. But how does one determine the optimum MAV trajectory? Especially when it is combined with the optimum trajectory of the low-thrust orbiter.

The proposed research would focus on the combined optimisation of an MAV trajectory and the trajectory of the low-thrust Mars 2022 orbiter. Also, one hypothesis is that great mass saving can be made if the orbiter and MAV would rendezvous within one single orbital revolution after MAV lift-off. Therefore, the question that should be answered is: what is the optimal trajectory solution for the combined trajectory problem of a high-thrust MAV and a low-thrust Mars orbiter performing a single-revolution rendezvous in Mars orbit? More information on the proposed topic is provided in **??**.

A mission such as MSR and the corresponding trajectories can be described in many different reference frames, or Frame of Reference (RF), and the motion of the MAV and the orbiter can be modelled in different ways. Therefore it is important to use the proper equations and environmental models. Also, the trajectory has to be determined or rather a prediction will have to be made. This can be done using integration methods. And finally, the optimum will have to be found using an optimisation method. All these different aspects are addressed in this literature study.

First however, it is important to determine the knowledge that already exists and the research that has already been performed. Therefore, **??** will describe previous sample return missions, low-thrust Spacecraft (s/c) missions, single-revolution rendezvous missions and the research performed in those fields. It will also describe the current MAV designs. Then before mathematically representing the problem it is important to understand in what kind of RF it has to be described. This will be done in **??**, followed by the MAV ascent and low-thrust Mars 2022 orbiter model descriptions in **????** respectively. Here, both chapters explain the assumptions and corresponding equations for each phase. One important aspect of the MAV ascent, which sets it apart from other sample return missions, is that Mars has an atmosphere which cannot be neglected. Accordingly **??** describes the different atmospheric models and the trade-off that was performed to decide which model to use in this thesis problem. Then the integration and optimisation are discussed in **??** and chapter 4 respectively. In the integrators chapter, different integration methods are described and a selection is made of the integration methods that will be used. The same is done for the different optimisers. All of this information will be used to define the final thesis topic, which is presented in **??**. For some of the aspects that will be treated in the final thesis problem, certain software is already available. A summary of this software is provided in **??**. Finally, a proposed schedule is presented in **??**, which shows the work which will have to be performed during the thesis work and the time that will have to be spend on each aspect of it. This literature study will serve as a guideline during the thesis project and provide background information for the final thesis report.

2

PROBLEM BACKGROUND

3

Models

4

OPTIMISATION

5

STANDARD INTEGRATION METHODS

6

TAYLOR SERIES INTEGRATION

7

PROGRAM OPTIMISATION TOOL

To perform the analysis associated with this thesis, a simulation and optimisation program is used. This optimisation tool is comprised of both existing (Section 7.1) and newly developed software (Section 7.2). It is written in C++ and is based on the Tudat structure. The purpose of the software is to simulate the trajectory of the MAV and optimise this trajectory with respect to the lowest propellant mass required. This tool is written such that the performance of Runge-Kutta-Fehlberg 4^{th} (5^{th}) order (RKF45) and TSI can be compared.

7.1. EXISTING SOFTWARE

The use of existing software can greatly improve the performance of the final tool and save time as well. Another important reason to use existing software is that this will make it easier for other people to use and incorporate into their software as well. The existing software used for this thesis is software that is currently being used by the space department of the TU Delft and (in case of SNOPT and Mars-GRAM) by the mission design section at JPL.

7.1.1. TUDAT

Tudat is, as the name suggests, a toolbox that can be used to solve numerous astrodynamic problems (Dirkx *et al.*, 2016). It was and still is being developed by students and staff of the Delft University of Technology. Specifically by the section Astrodynamics and Space missions of the Aerospace Engineering faculty. It is programmed in C++ and consists of a number of libraries. These libraries can be called upon by the user to invoke different Tudat functionalities such as standard reference frame transformations or often used integrators. The available software is completely validated and comes with its own tests to make sure that everything is working properly. It itself uses two external libraries: Eigen and Boost. Both these libraries will be discussed in Sections 7.1.2 and 7.1.3 respectively. In this thesis, the Tudat libraries are used for all standard mathematical and astrodynamic operations.

7.1.2. EIGEN

Eigen is an external C++ library that was written to perform linear algebra computations [1]. The software is free and easy to use, which is why it is widely used by the C++ community and thus also within Tudat (Dirkx *et al.*, 2016). Another advantage is that because it does not use any source files, it does not need to be build before using it. The Eigen libraries contain a number of standardized matrices and vectors, each with its own characteristics. An example of an often used vector is *Vector3d* (or *Eigen::Vector3d*), which can for instance be used to store the Cartesian position of a satellite. Here the *3* shows that it can store 3 values/parameters and the *d* shows that these are of the type *double*. It is mentioned on the Tudat wiki (Dirkx *et al.*, 2016) that these Eigen vectors and matrices should only be used if required for linear algebra computations. For ordinary storage, the C++ arrays, vectors and matrices should be used to save both storage and computation time.

[1] More documentation on Eigen can be found on eigen.tuxfamily.org/dox/ [Accessed 8 March 2016]

7.1.3. BOOST

Boost is a slightly more complicated set of C++ libraries, where compared to the Eigen library, Boost first has to be compiled before being able to use all of its functionalities. Fortunately, this compiling is performed by Tudat automatically when setting it up for the first time. Boost is described as an addition to the standard C++ libraries, thus adding more functionalities (Dirkx *et al.*, 2016) [2]. Within Tudat, Boost is used to pass free and class functions as an argument to another object and also for dynamic allocation using so-called pointers. Four libraries that are often used within Tudat are *boost::function*, *boost::bind*, *boost::shared_ptr* and *boost::make_shared*. The first two libraries are used to pass functions (a function is pointed to by *function* and called by *bind*) and the last two are used in case of dynamic allocation (*shared_ptr* is the pointer and *make_shared* is the object creator that returns a shared pointer to the created object).

7.1.4. PAGMO

PaGMO is a free optimisation tool developed by European Space Agency (ESA)s Advanced Concepts Team (ACT). It uses parallel computations to perform the optimisation and can even optimise for multi-objective problems. Parallel computation is the act of performing multiple computations on the same machine using different CPU cores. This allows the cost function to be computed for different sets of optimisation parameters at the same time and thus reducing the total CPU time required. However, this only works if the cost function evaluations are independent, which is not always the case (e.g. Dynamic Differential Evolution (DE) described by Qing (2009)). The tool itself incorporates many different local and global optimisation methods as mentioned by Izzo (2012), among which the optimisation method used in this thesis Monotonic Basin Hopping (MBH). This method has been written in PaGMO in such a way that it can use any of the provided local optimisers. PaGMO is written in C++ and requires the shared libraries of Boost to run [3]. Interfaces to external libraries are also provided, which can incorporate for instance SNOPT as a local optimisation method. In this thesis SNOPT is used as the local optimiser for MBH as implemented by PaGMO. More information on SNOPT is provided in Section 7.1.5. For SNOPT to be recognised by PaGMO, it has to be installed separately.

7.1.5. SNOPT

SNOPT was introduced by Gill *et al.* (2002) as a Sequential Quadratic Programming (SQP) method. It uses the first function derivatives and is very effective with highly constrained problems such as trajectory optimisation. Because it is based on SQP it is only able to find the local optimum and it is thus not guaranteed that this is also the global optimum. By combining SNOPT and MBH the global optimum can indeed be found or approached. The tool itself does not require that many evaluations, which is why it is very useful for complex problems with many optimisation variables (Gill *et al.*, 2008). The code for SNOPT has been written in Fortran, but can easily be translated to C,C++ using *f2c* which is provided with SNOPT as well [4]. This way it can be called by PaGMO. It should be noted that SNOPT is not free and can only be used under a licence agreement.

7.1.6. MARS-GRAM

Mars-GRAM is a high-fidelity atmospheric model developed by NASA to simulate the global atmospheric conditions on Mars (Justh and Justus, 2008) [5]. The model is based on NASA Ames Mars General Circulation Model (for altitudes between 0-80 km) and Mars Thermospheric General Circulation model (for altitude above 80 km). It can provide density, temperature and pressure data (among other data) with respect to the current altitude, latitude and longitude on Mars. Seasonal variations are taken into account in the model as well, which is why different calender dates will result in different atmospheric compositions. The tool can be used within a simulation tool or as a separate executable. Unfortunately, because it is so detailed, each computation requires a lot of CPU time. This is why it was decided to use the stand-alone Mars-GRAM executable to generate a detailed table with atmospheric data as a function of altitude, latitude and longitude at the start of the optimisation. Even generating this table required a lot of CPU time (on average a single computation using the stand-alone executable took 67.9 seconds to complete). The starting altitude was set at -0.6 km Mars Orbiter Laser Altimeter (MOLA) and advanced with a step-size of 0.1 km to 320 km altitude to cover the entire range that the MAV would have to cover. Also, the latitude and longitude were varied within

[2] More documentation on Boost can be found on http://www.boost.org/ [Accessed 8 March 2016]
[3] More documentation on PaGMO can be found on https://esa.github.io/pagmo/ [Accessed 9 March 2016]
[4] More documentation on SNOPT can be found on http://www.sbsi-sol-optimize.com/asp/sol_products_snopt_desc.htm [Accessed 9 March 2016]
[5] NASA website: https://see.msfc.nasa.gov/model-Marsgram [Accessed 9 March 2016]

10 degrees from the launch site with a step-size of 1 degree. A Matlab script was written to extract the relevant atmospheric data from the Mars-GRAM output files and write them into a .csv file, thus creating the required atmospheric data table. The atmospheric data in this table was then interpolated to provide an estimate of the atmospheric characteristics at every point along the ascent trajectory, which is required to compute the drag at each time step. Some of the earlier versions of Mars-GRAM are available for free (such as the Mars-GRAM 2005 version used in this thesis), however, the latests versions (such as the Mars-GRAM 2010 version used as a back-up in this thesis) require a licence agreement.

7.2. DEVELOPED SOFTWARE

This section of the software chapter describes the software that either had to be developed around existing software/libraries or had to be developed from scratch (the TSI propagator). Each piece of software is accompanied by the corresponding software architecture. Every next piece of software then indirectly incorporates the previous architecture through the use of the completed tool.

7.2.1. ATMOSPHERIC TABLE FUNCTION FIT

Using Mars-GRAM 2005, a table containing altitude, latitude and longitude dependent temperature and density data was produced. The altitude range was -0.6 to 320 km MOLA with a step-size of 0.01 km, the latitude and longitude ranges were centred around the launch site (21.0 °N and 74.5 °E) with a 10 degree range in each direction and a step-size of 1 degree. The rest of the input parameters were constant and can be seen in Appendix A. The temperature and density data produced is shown in Figures 7.1 and 7.2 respectively for 9 latitude and longitude combinations including the launch site itself.

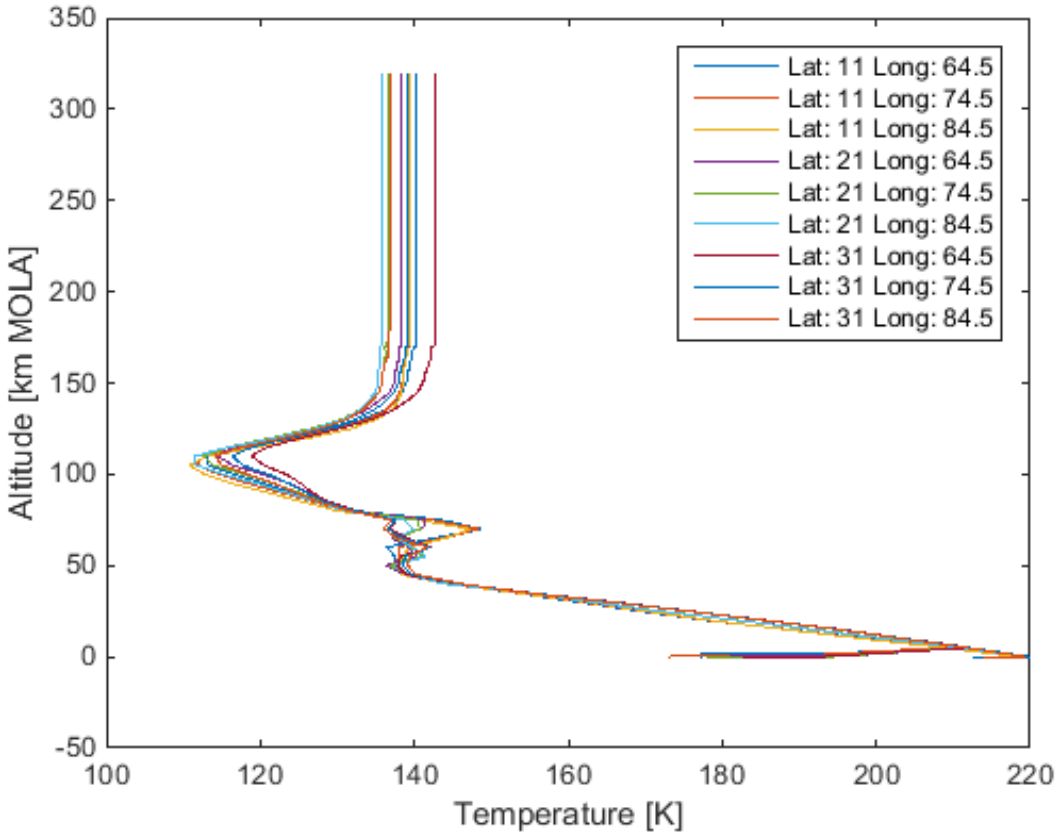

Figure 7.1: Temperature data generated with Mars-GRAM 2005 showing 9 different latitude and longitude combinations

Unfortunately discontinuous data tables cannot be used when integrating using TSI, which is why both these data tables had to be fitted with continuous functions. The temperature data could not be smoothly fit with one continuous function. Therefore, depending on the altitude range, a different approximation func-

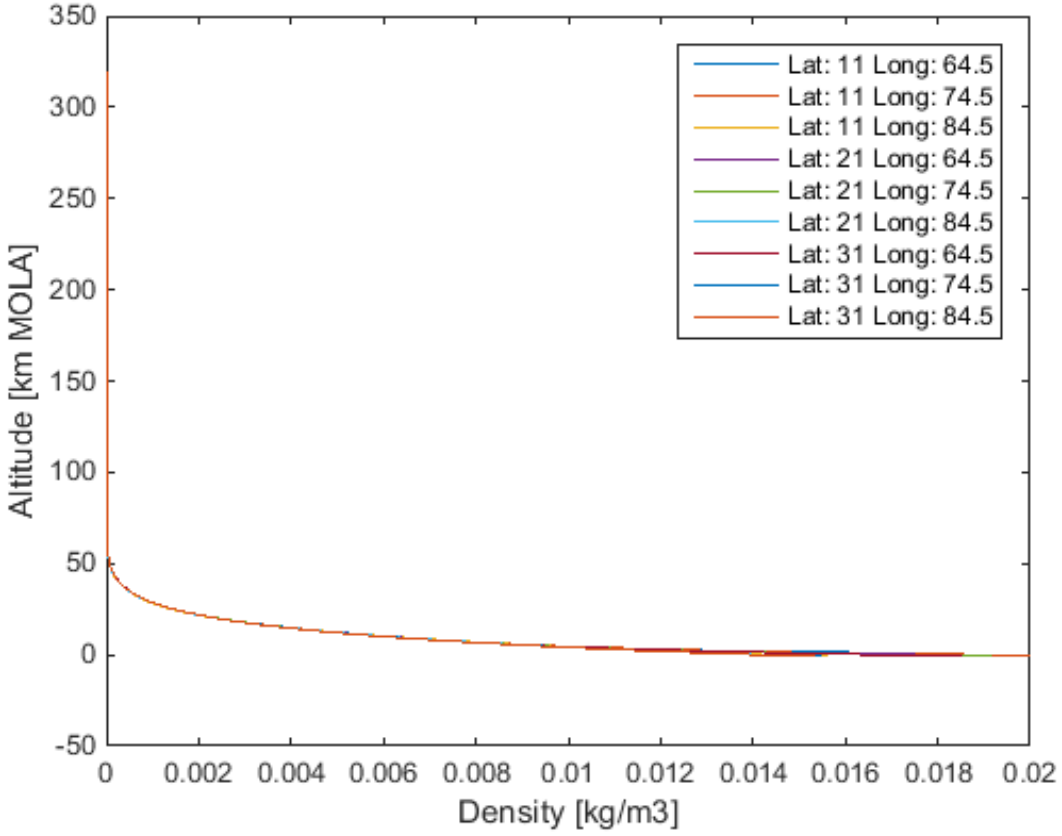

Figure 7.2: Density data generated with Mars-GRAM 2005 showing 9 different latitude and longitude combinations

tion is required. The condition to be met for a proper fit came from the differences in the temperature-altitude and density-altitude curves, where the maximum difference with respect to the launch site curve was taken. The requirement for the standard deviation of the polynomial curve fit was then to be (at least) one order lower than this maximum difference and that the maximum difference between the fit and the launch site curve was lower than the maximum difference. The temperature-altitude curve was split into 5 sections as roughly visualised in Figure 7.3. The number of sections come from both the shape of the curves and the requirement for accuracy and maximum order of the polynomial, which is set at 8 because otherwise the polynomial would get too long. Also, the number of sections were to be kept at a minimum.

Each section was fit with a polynomial function of the n^{th} order where the function is represented by Equation (7.1). The last section shows a constant temperature, thus the temperature of the launch site curve was chosen to represent this final section.

$$y = p_1 x^n + p_2 x^{n-1} + \cdots + p_n x + p_{n+1} \tag{7.1}$$

A lower order is preferred, because then the fitted function will be simpler to evaluate and contain fewer terms. However the order has to be high enough to meet the accuracy requirements. Table 7.1 shows the orders that were required and the deviations to the launch site temperature-altitude curve. The actual corresponding parameters are provided in Table 7.2. It should be noted that the first few temperature data values were so different from the rest of the curve that it was assumed that this is a lack of the Mars-GRAM program and were thus treated as outliers.

The complete polynomial fit for the launch site curve for the temperature is shown in Figure 7.4.

The density fit was slightly more difficult because the curves are all very similar and thus result in a higher accuracy requirement for the fit. At first glance it looks like a natural logarithmic function, unfortunately an ordinary exponential did not fit the curve. This is why a more extensive exponential fit was required. The natural logarithm of the data has been plotted in Figure 7.5.

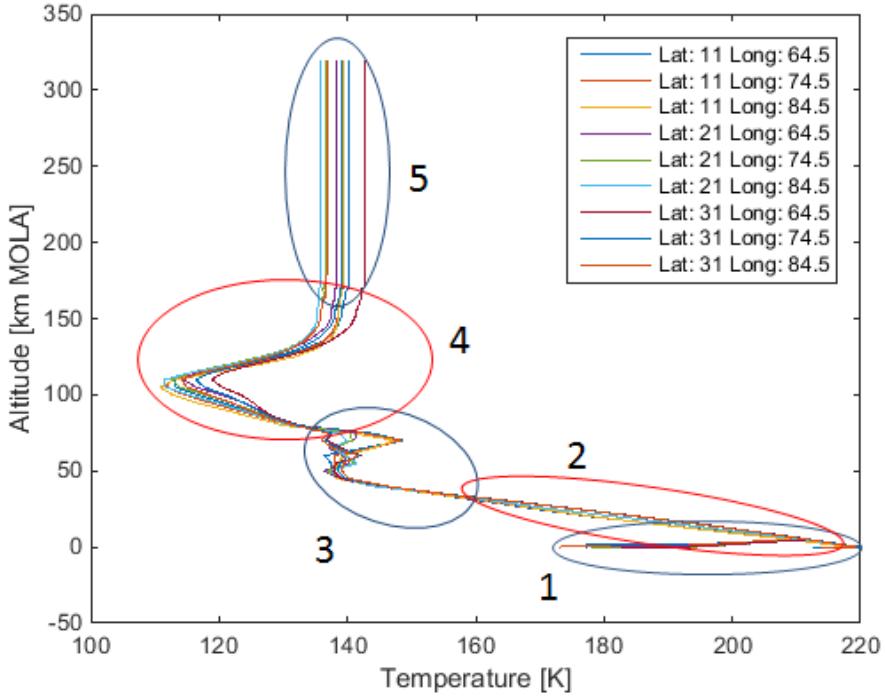

Figure 7.3: Different temperature curve sections

Table 7.1: Temperature curve fit data

Section	**Altitude range [km MOLA]**	**Order**	**Maximum standard deviation [K]**	**Maximum curves difference [K]**	**Maximum difference with launch site curve [K]**
1	-0.6 to 5.04	1	0.0312	25.8	0.177
2	5.04 to 35.53	2	0.287	3.90	0.7056
3	35.53 to 75.07	6	0.624	8.00	1.69
4	75.07 to 170.05	8	0.523	6.60	2.45

Table 7.2: Temperature curve fit parameters (rounded to 3 decimal points)

Section	$\mathbf{p_1}$	$\mathbf{p_2}$	$\mathbf{p_3}$	$\mathbf{p_4}$	$\mathbf{p_5}$	$\mathbf{p_6}$	$\mathbf{p_7}$	$\mathbf{p_8}$	$\mathbf{p_9}$
1	3.415	194.165							
2	0.006	-2.130	222.052						
3	-5.388 $\cdot 10^{-7}$	1.785 $\cdot 10^{-4}$	-0.0243	1.733	-68.294	1.407 $\cdot 10^{3}$	-1.167 $\cdot 10^{4}$		
4	4.1942 $\cdot 10^{-12}$	-4.328 $\cdot 10^{-9}$	1.931 $\cdot 10^{-6}$	-4.862 $\cdot 10^{-4}$	0.076	-7.405	447.378	-1.523 $\cdot 10^{4}$	2.236 $\cdot 10^{5}$

With the data represented in the logarithmic domain, again a polynomial function can be fit. The total fit would then satisfy Equation (7.2).

$$y = exp\left(p_1 x^n + p_2 x^{n-1} + \cdots + p_n x + p_{n+1}\right) \tag{7.2}$$

The same polynomial requirements as for the temperature curve were enforced for the density curve as well. However, because the polynomial is used in an exponential, some extra requirements are needed to

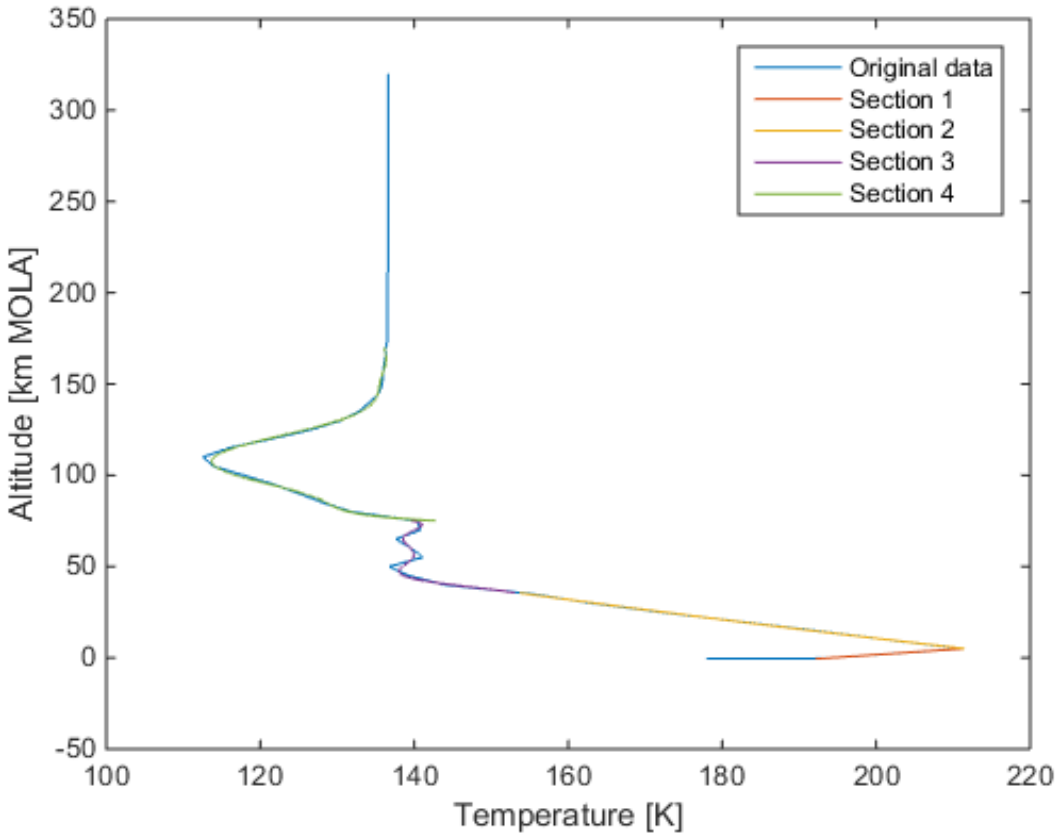

Figure 7.4: All section fits for the launch site temperature data curve

assure the accuracy of the fit. One requirement is that the maximum difference between the final exponential fit and the normal launch site density curve is smaller than the maximum difference between all the data curves. Also, in this case the standard deviation of the difference between the exponential fit and the normal launch site curve had to be within the range of standard deviations of the difference between the different data curves. This meant that even though an 8^{th} order polynomial fit could be achieved for the natural logarithmic data with the required accuracy, when converted to the exponential fit, the last two requirements were not met. Before it was mentioned that an order higher than 8 was not desirable. However, in this case, a single exponential fit could be achieved using a 10^{th} order polynomial. This fit meant that the density curve did not have to be split up at all, which makes the integration slightly easier. Therefore, it was decided that a 10^{th} order polynomial was acceptable in this case. The results of the fit is presented in Tables 7.3 and 7.4 and the exponential fit curve is shown in Figure 7.6.

Table 7.3: Density curve fit data (10^{th} order polynomial)

Maximum polynomial standard deviation [kg/m^3]	0.0501
Maximum natural logarithmic curves difference [kg/m^3]	0.460
Maximum polynomial difference with natural logarithmic launch site curve [kg/m^3]	0.160
Maximum curves difference [kg/m^3]	$3.910 \cdot 10^{-3}$
Maximum difference with launch site curve [kg/m^3]	$2.826 \cdot 10^{-3}$
Maximum standard deviation curves difference [kg/m^3]	$2.106 \cdot 10^{-4}$
Standard deviation exponential fit difference [kg/m^3]	$1.167 \cdot 10^{-4}$

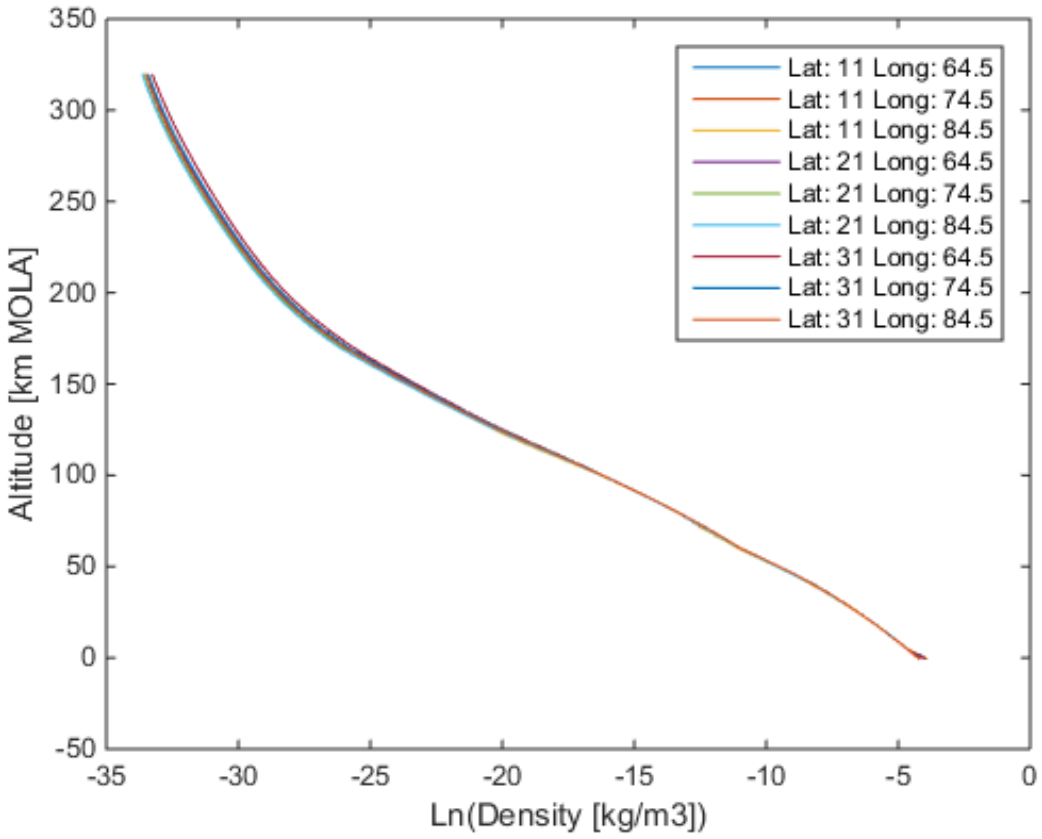

Figure 7.5: Natural logarithmic plot of the density data

Table 7.4: Density curve fit parameters (rounded to 3 decimal points)

p_1	p_2	p_3	p_4	p_5	p_6	p_7	p_8	p_9	p_{10}	p_{11}
$2.287 \cdot 10^{-21}$	$-3.724 \cdot 10^{-18}$	$2.559 \cdot 10^{-15}$	$-9.620 \cdot 10^{-13}$	$2.146 \cdot 10^{-10}$	$-2.884 \cdot 10^{-8}$	$2.273 \cdot 10^{-6}$	$-9.604 \cdot 10^{-5}$	$1.414 \cdot 10^{-3}$	-0.0962	-4.172

7.2.2. DRAG COEFFICIENT GRAPH FUNCTION FIT

Similar to the temperature and density curves, the relation between Mach number and drag coefficient, as depicted in Figure 7.7, had to be modelled as a continuous function as well. Again, it could not be fitted using one continuous function, but instead had to be modelled by different functions.

Fortunately, this curve is already an approximation and thus consists of linear elements only. It can be split up into 6 different sections where the first and last section are constant. Using a similar polynomial fit as before, but now for 1 order only, a linear fit could be made for each of the remaining 4 sections. The corresponding parameters are shown in Table 7.5 and the curve fit is shown in Figure 7.8.

Table 7.5: Drag coefficient curve fit parameters (rounded to 3 decimal points)

Section	p_1	p_2
1	0.400	$-2.483 \cdot 10^{-16}$
2	0.567	-0.167
3	-0.142	0.754
4	-0.0667	0.567

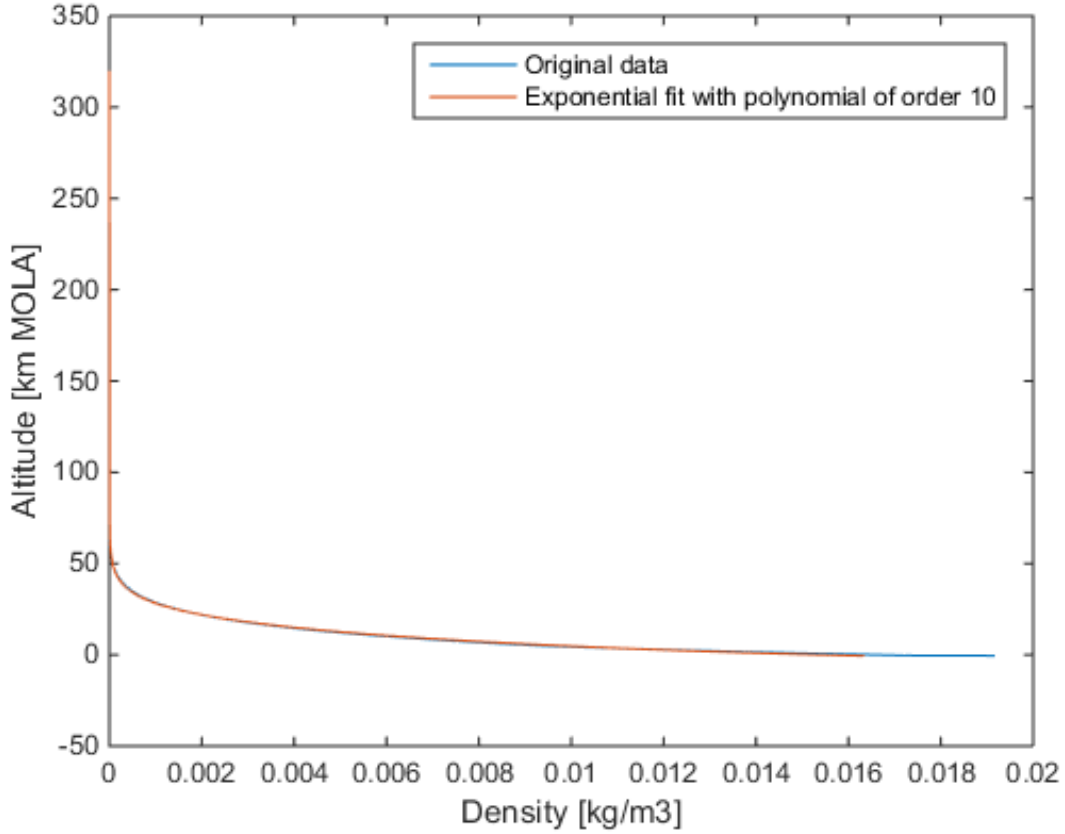

Figure 7.6: All section fits for the launch site density data curve

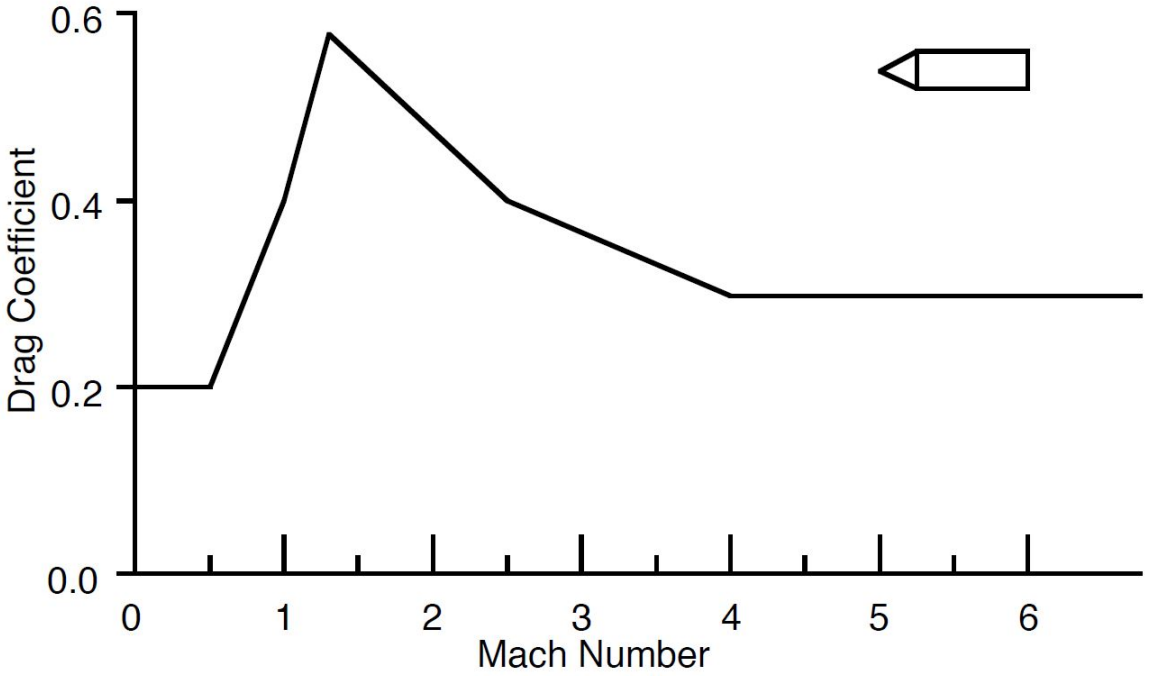

Figure 7.7: Drag coefficient as a function of Mach number Whitehead (2004)

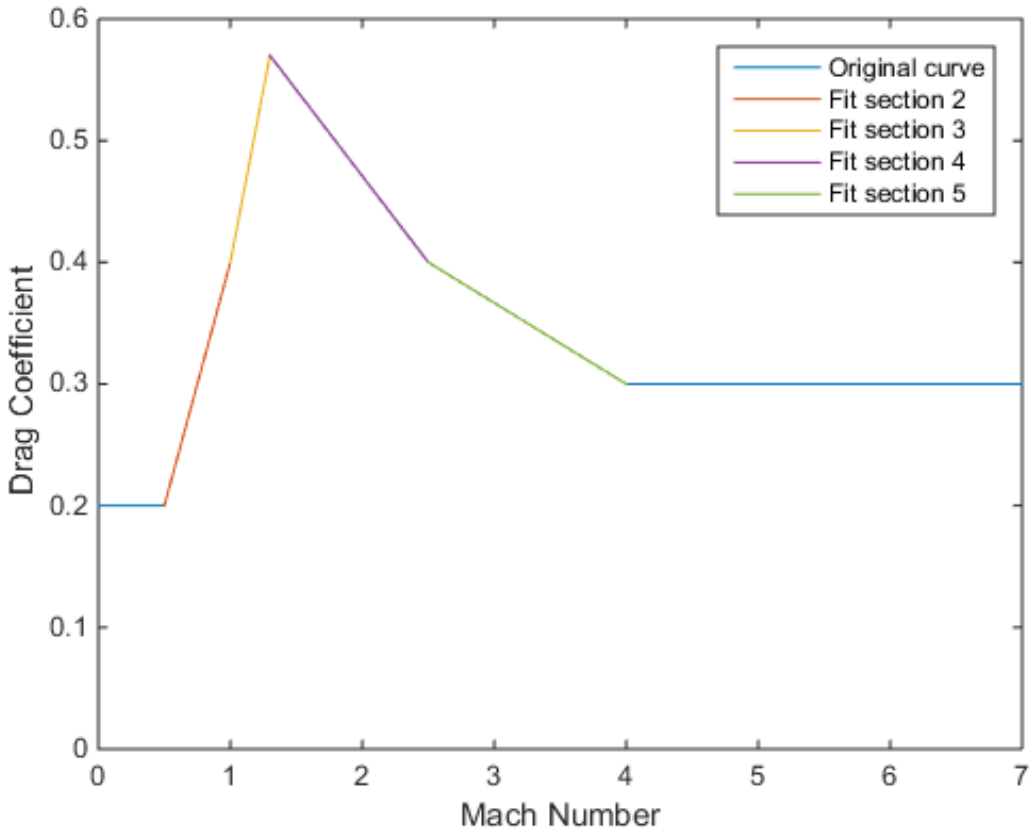

Figure 7.8: All section fits for the drag coefficient - Mach curve

7.2.3. RK4 AND RKF PROPAGATOR

The RK4 and RKF (or traditional) propagator architecture is described in Figure 7.9. It starts with the current state, which is then passed on to the state derivative function. The state derivative function is used by the RK4 and RKF integrators to determine the next state by calling the function a number of times depending on the used method. Both RK4 and RKF45 (and higher order RKF integrators) are already available through the Tudat libraries. RK4 can be called by including the rungeKutta4Integrator.h header file, and RKF45 can be called by including the rungeKuttaVariableStepSizeIntegrator.h header file. This integration process is repeated until the final condition is met. Within the state derivative function all the sate derivatives are updated and stored. The current position is used to update the gravitational acceleration on the MAV, the current mass is used to determine the accelerations caused by the thrust and finally the complete state is required to determine the accelerations caused by the drag. Both the drag and thrust accelerations have to be transformed to the inertial frame using the updated angles from the current state. The function also computes the current mass flow rate, however since the thrust is constant, this does not change over time. In the state derivative function, all the transformations are governed by pre-developed functions within the Tudat library, which includes the state transformations and the frame transformation from the body frame to the inertial frame. The transformation from the propulsion frame to the body frame is however not included in Tudat and had to be written.

7.2.4. TSI PROPAGATOR

The TSI propagator has a significantly different architecture compared to the traditional propagator as can be seen in Figure 7.10. TSI requires an initial order and step-size to start the integration process. In this thesis it has been decided to keep the order the same throughout the entire integration. The step-size will change during the integration depending on the Taylor series evaluations. The initial state is set as the current state and is fed into the TSI block. Within this block, first the auxiliary equations and functions are called, which were set-up for this particular problem. They are evaluated using the current state. These auxiliary equations and functions already include all the reference frame and coordinate transformations, as well as approximate

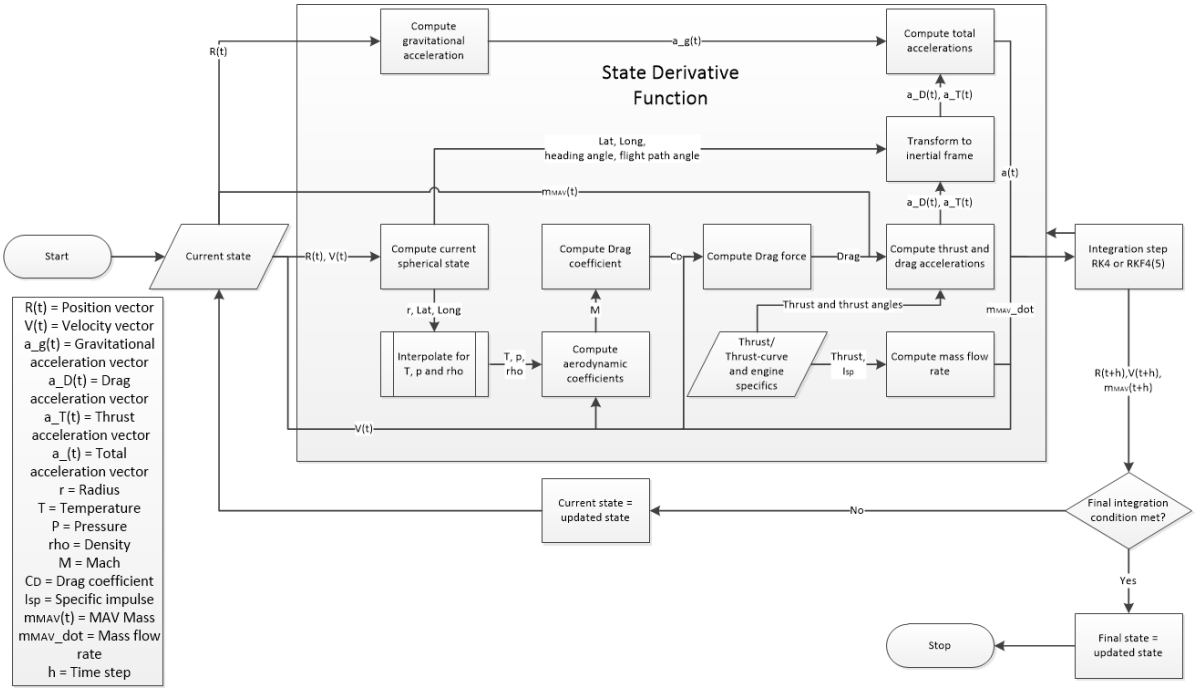

Figure 7.9: RK4 and RKF interface architecture

atmospheric parameter functions. This is required to set-up the recurrence relations, which is where TSI differs from the traditional propagator. Once the auxiliary equations and functions have been computed they are used to compute the Taylor coefficients through the recurrence relations set-up for the thesis problem. These coefficients are then stored for later use and are also passed to the block creating the Taylor series expansion for every state variable thus creating he updated state. The last two coefficients are then used to determine the next step-size. This continues until the final integration condition has been met.

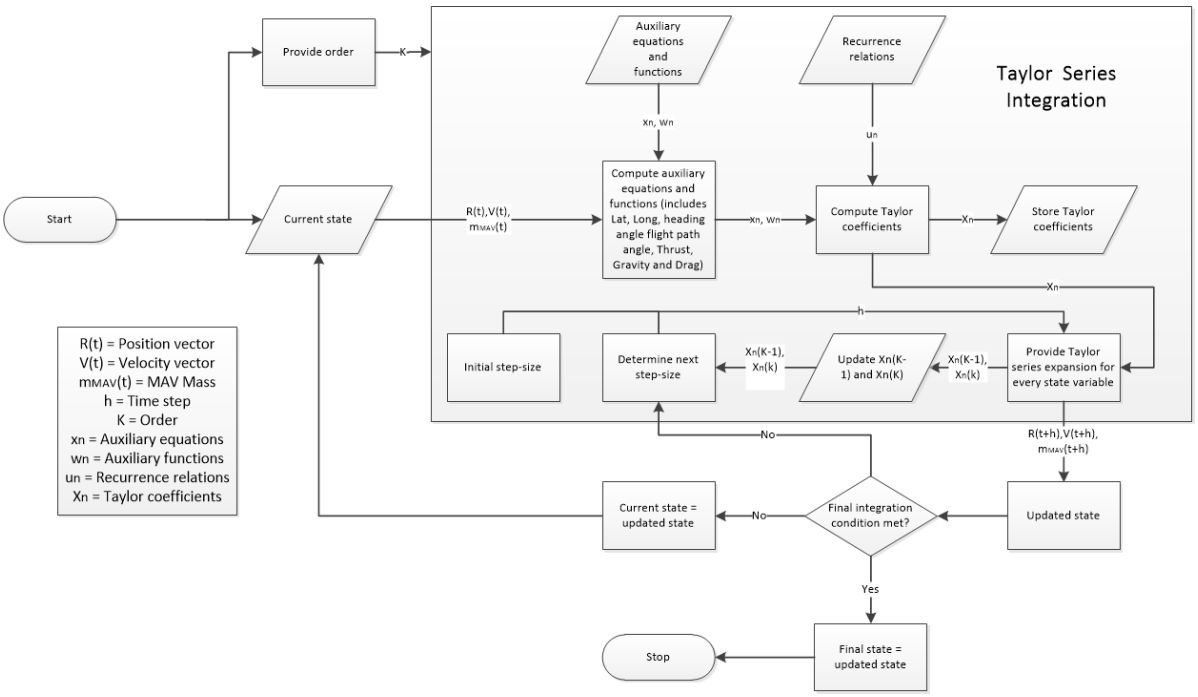

Figure 7.10: TSI architecture

7.2.5. OPTIMISER

The optimisation software is a combination of the SNOPT local optimisation tool and PaGMOs MBH. Even though both these tools were already available and did not have to be developed, it is still important to understand how the rest of the software interacts with the optimiser. This is why Figure 7.11 shows the architecture of the MBH optimiser. It starts with the initial generation of the optimisation parameters, after which the 'Number of not improved iterations' is set to zero. This is then fed into the local optimiser, where the trajectory is integrated using the previously described tools. Once a local optimised trajectory is found, it is stored if it is better than the previous local optimised trajectory and the counter is set to zero again. If the newly found trajectory is not better than the current best the 'Number of not improved iterations' is increased by one. Once the maximum number of not improved iterations is met, the current best optimal trajectory (which is the optimum for the current "funnel") is stored and the process is repeated till the final global optimisation condition is met. At this point the global optimum is the best optimal trajectory from all the funnels computed at that time, which is then returned as the program solution.

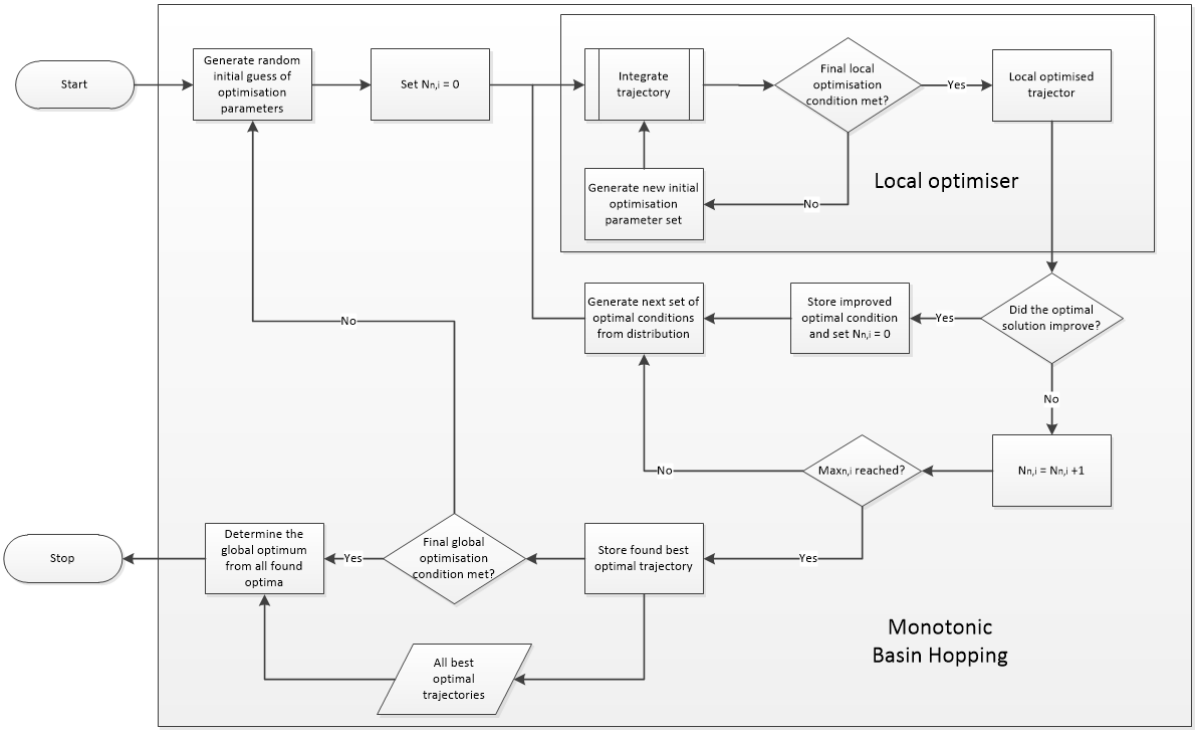

Figure 7.11: Optimiser interface architecture

8

VERIFICATION AND VALIDATION

Verification is the process of determining whether a program meets the requirements or not. Is it working the way it is suppose to? As soon as it works and produces output (verified), these outputs can be compared to other data from which it is known that it is correct. This is called validation. If a program is verified and validated, the outputs should be correct. Fortunately, all the existing software has already been validated, which means that they only still have to be verified to make sure everything is working properly on the simulation computer. Each of the software packages comes with tests which can be used to do this. If all the tests are passed it means that the software is verified for the computer and ready to use. Since Tudat shipped with Eigen and Boost, the Tudat test files also test these libraries. All the test were passed, which means that the Tudat, Eigen and Boost libraries are working properly. However, because the integrators are an intricate part of this thesis, it was decided to perform a separate verification for the Runge-Kutta integrators. This verification is described in Section 8.2. Similarly, PaGMO was verified using its test files.

SNOPT verification still has to be done and added!! As soon as I can get it to work....

Mars-GRAM also came with its own verification test, where three delivered output files had to be replicated. These verification tests were also successful.

8.1. INTERPOLATION

8.2. RK4 AND RKF INTEGRATORS

The tutorial page of the Tudat website (Dirkx *et al.*, 2016) offers two integration tutorials: one involving RK4 and another involving variable step-size Runge-Kutta methods including RKF45. The objective of the tutorial is to get familiar with the different integration methods available in Tudat. At the same time, a small data table has to be reproduced, which also serves as a verification test. For each of the integrators the same problem was addressed: the computation of the velocity of a falling body after a certain amount of time assuming no drag. The results that had to be reproduced are presented in Table 8.1.

Table 8.1: Verification data for the standard integrators

End time [s]	1.0	5.0	15.0	25.0
Velocity [m/s]	-9.81	-49.05	-147.15	-245.25

The first script was written using the instructions from the tutorial and is called numericalintegrators.cpp. This script uses the rungeKutta4Integrator.h header file and the RungeKutta4IntegratorXd function from this header file. This function requires three inputs: the state derivative function (problem specific), en initial time and the initial state. Using the .integrateTo extension the end state at a certain end time can be integrated. This requires the end time and the step-size. For RK4 the step-size is constant. This resulted in the same values as presented in Table 8.1.

The second script was used to test the variable step-size integrators (including RKF45). This script is called rungekuttavariable.cpp and uses the rungeKuttaCoefficients.h and rungeKuttaVariableStepSizeIntegrator.h

header files. In this case the RungeKuttaVariableStepSizeIntegratorXd function was used which requires the Runge-Kutta coefficients (from the respective header file), the state derivative function, the initial time, initial state, zero minimum step-size, infinite maximum step-size, relative tolerance and the absolute tolerance as inputs. In this case the integration can be done in individual steps using .performIntegrationStep with the current step-size as the only input. The current step-size is computed in the integration method itself, but in this case it is checked to make sure that the step-size does not take the function beyond the specified end time. The integration steps are then repeated until the end time is met. Running this script resulted in the same results as presented in Table 8.1 as well.

These results, combined with the fact that the test files for these two methods produced no errors is proof that they are working accordingly. Thus it can be said that the standard integration methods used in this thesis are verified and ready for use in the optimisation tool. However, during the development of the trajectory propagation tools, the entire tool (including the integrators) will be verified again to determine the performance of the integrators.

8.3. TAYLOR SERIES INTEGRATION
8.4. COMPLETE TRAJECTORY PROPAGATION
8.5. OPTIMISER
8.6. COMPLETE OPTIMISATION TOOL

9

RESULTS

10

Analysis

11

CONCLUSIONS AND RECOMMENDATIONS

A

MARS-GRAM 2005 INPUT FILE

```
$INPUT
 LSTFL    = 'LIST.txt'
 OUTFL    = 'OUTPUT.txt'
 TRAJFL   = 'TRAJDATA.txt'
 profile  = 'null'
 WaveFile = 'null'
 DATADIR  = '/home/stachap/MarsGram/binFiles_TUDelft/'
 GCMDIR   = '/home/stachap/MarsGram/binFiles_TUDelft/'
 IERT     = 0
 IUTC     = 1
 MONTH    = 2
 MDAY     = 28
 MYEAR    = 2025
 NPOS     = 32061
 IHR      = 12
 IMIN     = 0
 SEC      = 0.0
 LonEW    = 1
 Dusttau  = 0
 Dustmin  = 0.3
 Dustmax  = 1.0
 Dustnu   = 0.003
 Dustdiam = 5.0
 Dustdens = 3000.
 ALS0     = 0.0
 ALSDUR   = 48.
 INTENS   = 0.0
 RADMAX   = 0.0
 DUSTLAT  = 0.0
 DUSTLON  = 0.0
 MapYear  = 0
 F107     = 68.0
 STDL     = 0.0
 NR1      = 1234
 NVARX    = 1
 NVARY    = 0
 LOGSCALE = 0
 FLAT     = 21
 FLON     = 74.5
 FHGT     = -0.6
 MOLAhgts = 1
 hgtasfcm = 0.
 zoffset  = 0.
 ibougher = 0
 DELHGT   = 0.01
 DELLAT   = 0.0
 DELLON   = 0.0
 DELTIME  = 0.0
 ΔTEX = 0.0
```

```
 profnear = 0.0
 proffar  = 0.0
 rpscale  = 1.0
 rwscale  = 1.0
 wlscale  = 1.0
 wmscale  = 0.0
 blwinfac = 0.0
 NMONTE   = 1
 iup      = 13
 WaveA0   = 1.0
 WaveDate = 0.0
 WaveA1   = 0.0
 Wavephi1 = 0.0
 phi1dot  = 0.0
 WaveA2   = 0.0
 Wavephi2 = 0.0
 phi2dot  = 0.0
 WaveA3   = 0.0
 Wavephi3 = 0.0
 phi3dot  = 0.0
 iuwave   = 0
 Wscale   = 20.
 corlmin  = 0.0
 ipclat   = 1
 requa    = 3396.19
 rpole    = 3376.20
 idaydata = 1
$END

Explanation of variables:
LSTFL    =  List file name (CON for console listing)
OUTFL    =  Output file name
TRAJFL   =  (Optional) Trajectory input file. File contains time (sec)
             relative to start time, height (km), latitude (deg),
             longitude (deg W if LonEW=0, deg E if LonEW=1, see below)
profile  =  (Optional) auxiliary profile input file name
WaveFile =  (Optional) file for time-dependent wave coefficient data.
             See file description under parameter iuwave, below.
DATADIR  =  Directory for COSPAR data and topographic height data
GCMDIR   =  Directory for GCM binary data files
IERT     =  1 for time input as Earth-Receive time (ERT) or 0 Mars-event
             time (MET)
IUTC     =  1 for time input as Coordinated Universal Time (UTC), or 0
             for Terrestrial (Dynamical) Time (TT)
MONTH    =  (Integer) month of year
MDAY     =  (Integer) day of month
MYEAR    =  (Integer) year (4-digit; 1970-2069 can be 2-digit)
NPOS     =  max # positions to evaluate (0 = read data from trajectory
             input file)
IHR      =  Hour of day (ERT or MET, controlled by IERT and UTC or TT,
             controlled by IUTC)
IMIN     =  minute of hour (meaning controlled by IERT and IUTC)
SEC      =  seconds of minute (meaning controlled by IERT and IUTC).
             IHR:IMIN:SEC is time for initial position to be evaluated
LonEW    =  0 for input and output West longitudes positive; 1 for East
             longitudes positive
Dusttau  =  Optical depth of background dust level (no time-developing
             dust storm, just uniformly mixed dust), 0.1 to 3.0, or use
             0 for assumed seasonal variation of background dust
Dustmin  =  Minimum seasonal dust tau if input Dusttau=0 (≥0.1)
Dustmax  =  Maximum seasonal dust tau if input Dusttau=0 (≤1.0)
Dustnu   =  Parameter for vertical distribution of dust density (Haberle
             et al., J. Geophys. Res., 104, 8957, 1999)
Dustdiam =  Dust particle diameter (micrometers, assumed monodisperse)
Dustdens =  Dust particle density (kg/m**3)
ALS0     =  starting Ls value (degrees) for dust storm (0 = none)
ALSDUR   =  duration (in Ls degrees) for dust storm (default = 48)
INTENS   =  dust storm intensity (0.0 - 3.0)
RADMAX   =  max. radius (km) of dust storm (0 or >10000 = global)
DUSTLAT  =  Latitude (degrees) for center of dust storm
DUSTLON  =  Longitude (degrees) (West positive if LonEW=0, or East
```

```
                positive if LonEW = 1) for center of dust storm
MapYear   =  1 or 2 for TES mapping year 1 or 2 GCM input data, or 0 for
                Mars-GRAM 2001 GCM input data sets
F107      =  10.7 cm solar flux (10**-22 W/cm**2 at 1 AU)
NR1       =  starting random number (0 < NR1 < 30000)
NVARX     =  x-code for plotable output (1=hgt above MOLA areoid).
                See file xycodes.txt
NVARY     =  y-code for 3-D plotable output (0 for 2-D plots)
LOGSCALE  =  0=regular SI units, 1=log-base-10 scale, 2=percentage
                deviations from COSPAR model, 3=SI units, with density
                in kg/km**3 (suitable for high altitudes)
FLAT      =  initial latitude (N positive), degrees
FLON      =  initial longitude (West positive if LowEW = 0 or East
                positive if LonEW = 1), degrees
FHGT      =  initial height (km); ≤-10 means evaluate at surface height;
                > 3000 km means planeto-centric radius
MOLAhgts  =  1 for input heights relative to MOLA areoid, otherwise
               input heights are relative to reference ellipsoid
hgtasfcm  =  height above surface (0-4500 m); use if FHGT ≤ -10. km
zoffset   =  constant height offset (km) for MTGCM data or constant
               part of Ls-dependent (Bougher) height offset (0.0 means
               no constant offset).  Positive offset increases density,
               negative offset decreases density.
ibougher  =  0 for no Ls-dependent (Bougher) height offset term; 1
               means add Ls-dependent (Bougher) term, -A*Sin(Ls) (km),
               to constant term (zoffset) [offset amplitude A = 2.5 for
               MapYear=0 or 0.5 for MapYear > 0]; 2 means use global mean
               height offset from data file hgtoffst.dat; 3 means use
               daily average height offset at local position; 4 means
               use height offset at current time and local position.
               Value of zoffset is ignored if ibougher = 2, 3, or 4.
DELHGT    =  height increment (km) between steps
DELLAT    =  Latitude increment (deg) between steps (Northward positive)
DELLON    =  Longitude increment (deg) between steps (Westward positive
                if LonEW = 0, Eastward positive if LonEW = 1)
DELTIME   =  time increment (sec) between steps
ΔTEX =  adjustment for exospheric temperature (K)
profnear  =  Lat-lon radius (degrees) within which weight for auxiliary
                profile is 1.0 (Use profnear = 0.0 for no profile input)
proffar   =  Lat-lon radius (degrees) beyond which weight for auxiliary
                profile is 0.0
rpscale   =  random density perturbation scale factor (0-2)
rwscale   =  random wind perturbation scale factor (≥0)
wlscale   =  scale factor for perturbation wavelengths (0.1-10)
wmscale   =  scale factor for mean winds
blwinfac  =  scale factor for boundary layer slope winds (0 = none)
NMONTE    =  number of Monte Carlo runs
iup       =  0 for no LIST and graphics output, or unit number for output
WaveA0    =  Mean term of longitude-dependent wave multiplier for density
WaveDate  =  Julian date for (primary) peak(s) of wave (0 for no traveling
                component)
WaveA1    =  Amplitude of wave-1 component of longitude-dependent wave
                multiplier for density
Wavephi1  =  Phase of wave-1 component of longitude-dependent wave
                multiplier (longitude, with West positive if LonEW = 0,
                East positive if LonEW = 1)
phi1dot   =  Rate of longitude movement (degrees per day) for wave-1
                component (Westward positive if LonEW = 0, Eastward
                positive if LonEW = 1)
WaveA2    =  Amplitude of wave-2 component of longitude-dependent wave
                multiplier for density
Wavephi2  =  Phase of wave-2 component of longitude-dependent wave
                multiplier (longitude, with West positive if LonEW = 0,
                East positive if LonEW = 1)
phi2dot   =  Rate of longitude movement (degrees per day) for wave-2
                component (Westward positive if LonEW = 0, Eastward
                positive if LonEW = 1)
WaveA3    =  Amplitude of wave-3 component of longitude-dependent wave
                multiplier for density
Wavephi3  =  Phase of wave-3 component of longitude-dependent wave
                multiplier (longitude, with West positive if LonEW = 0,
```

```
192                East positive if LonEW = 1)
193   phi3dot   =  Rate of longitude movement (degrees per day) for wave-3
194                component (Westward positive if LonEW = 0, Eastward
195                positive if LonEW = 1)
196   iuwave    =  Unit number for (Optional) time-dependent wave coefficient
197                data file "WaveFile" (or 0 for none).
198                WaveFile contains time (sec) relative to start time, and
199                wave model coefficients (WaveA0 thru Wavephi3) from the
200                given time to the next time in the data file.
201   Wscale    =  Vertical scale (km) of longitude-dependent wave damping
202                at altitudes below 100 km (10≤Wscale≤10,000 km)
203   corlmin   =  minimum relative step size for perturbation updates
204                (0.0-1.0); 0.0 means always update perturbations, x.x
205                means only update perturbations when corlim > x.x
206   ipclat    =  1 for Planeto-centric latitude and height input,
207                0 for Planeto-graphic latitude and height input
208   requa     =  Equatorial radius (km) for reference ellipsoid
209   rpole     =  Polar radius (km) for reference ellipsoid
210   idaydata  =  1 for daily max/min data output; 0 for none
```

BIBLIOGRAPHY

D. Dirkx, K. Kumar, E. Doornbos, E. Mooij, and R. Noomen, *TUDAT,* (2016), [online database], URL: tudat.tudelft.nl [cited 8 March 2016] (Only available from the TU Delft network).

A. Qing, *Differential Evolution: Fundamentals and Applications in Electrical Engineering* (John Wiley & Sons, Singapore, 2009).

D. Izzo, *PyGMO and PyKEP: Open source tools for massively parallel optimization in astrodynamics (the case of interplanetary trajectory optimization),* in *5th International Conference on Astrodynamics Tools and Techniques (ICATT)* (2012).

P. E. Gill, W. Murray, and M. A. Saunders, *SNOPT: An SQP algorithm for large-scale constrained optimization,* in *SIAM journal on optimization*, Vol. 12 (2002) pp. 979–1006.

P. E. Gill, W. Murray, and M. A. Saunders, *Users guide for SNOPT version 7 - Software for large-scale nonlinear programming,* (2008), [online database], URL: http://web.stanford.edu/group/SOL/guides/ [cited 23 October 2015].

H. L. Justh and C. G. Justus, *Utilizing Mars global reference atmospheric model (Mars-GRAM 2005) to evaluate entry probe mission sites,* Presentation (2008), [online database], URL: https://smartech.gatech.edu/bitstream/handle/1853/26375/34-186-1-PB.pdf?sequence=1 [cited 10 December 2015].

J. C. Whitehead, *Mars Ascent Propulsion Trades with Trajectory Analysis,* in *40th AIAA/ASME/SAE/ASEE Joint Propulsion Conference and Exhibit* (American Institude of Aeronautics and Astronautics, 2004).