Mars Sample Return ascent trajectory analysis 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

NOMENCLATURE

Greek

η	Numerical approximation	-
η	Step multiplication factor	-
τ	Local error tolerance	-
Φ	Increment function	-

Roman

h	(Current) step-size	s
t_0	Initial time	s
$T_{n,k}$	Truncation error	-
$\mathbf{x}_i$	Current data point	-
$\mathbf{x}_{i+1}$	Next data point	-
x_n	nth variable	-
$\mathbf{X}$	State vector	-

ABBREVIATIONS

AB4	Adams-Bashforth 4^{th} order
AB6	Adams-Bashforth 6^{th} order
ABM4	Adams-Bashforth-Moulton 4^{th} order
ABM12	Adams-Bashforth-Moulton 12^{th} order
DLR	German Aerospace Center (Deutsches Zentrum für Luft- und Raumfahrt)
DOPRIN87	Dormand-Prince 8^{th} (7^{th}) order
ESA	European Space Agency
FPA	Flight-path angle
GLOM	Gross Lift-Off Mass
GRAM	Global Reference Atmospheric Model
JPL	Jet Propulsion Laboratory
MAV	Mars Ascent Vehicle
MOLA	Mars Orbiter Laser Altimeter
MSR	Mars Sample Return
RHR	Right-hand-rule
RKF45	Runge-Kutta-Fehlberg 4^{th} (5^{th}) order
RKF56	Runge-Kutta-Fehlberg 5^{th} (6^{th}) order
RKF78	Runge-Kutta-Fehlberg 7^{th} (8^{th}) order
RKF	Runge-Kutta-Fehlberg
RK4	Runge-Kutta 4^{th} order
RKN12	Runge-Kutta-Nyström 12^{th} order
s/c	Spacecraft
SC14	Störmer-Cowell 14^{th} order
SG	Shampine-Gordon
SNOPT	Sparse Nonlinear Optimizer
SSTO	Single Stage to Orbit
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 carried out. 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 ascent trajectory of the Mars Ascent Vehicle (MAV) responsible for transporting the dirt and soil samples into a Martian orbit. Chapter 2 explains the reasoning behind focusing on the ascent trajectory of the MSR as opposed to many other research topics that were available. In the field of ascent trajectory simulation, standard integrators such as Runge-Kutta-Fehlberg (RKF) are often used to simulate the trajectory propagation. However, in recent years, another integration method called Taylor Series integration (TSI) has shown promising improvements compared to RKF in the field of both orbit trajectory analysis as well as (re-)entry cases. This is why TSI is the focus of this thesis research. The primary research question is: "Does the Taylor Series integration method show similar performance improvements over Runge-Kutta-Fehlberg for the Mars ascent case as was observed for the orbital trajectories and (re-)entry cases". Chapter 2 also discusses some of the early assumptions, and the required reference systems.
To be able to simulate the launch trajectory of the MAV, real life flight conditions have to be modelled in such a way that the results approximate the real launch. The models used in this research are described in Chapter 3 and are chosen such that the problem is not too complex but at the same time still represents the real life scenario well enough to be able to draw meaningful conclusions.
Many different standard integrators are available to be used in such a problem and a selection of those are described in Chapter 4. However, out of those methods, the best comparison integration methods turned out to be RKF methods. And from the available RKF methods, RKF78 was chosen for the final comparison with TSI because it showed the best performance in the ascent case.
But because the RKF78 method is a standard method that is widely available, the main focus and efforts of this research is put on the TSI method. In Chapter 5 the TSI is explained in detail and three different application methods are described: a normal Cartesian method, a unit vector Cartesian method and a Spherical method. Eventually it was opted to use the Spherical method.
Both RKF78 and TSI had to be incorporated into a program simulation tool which is capable of computing the launch trajectory of the MAV given a number of initial conditions. The complete set-up of the simulation tool is described in Chapter 6 and includes both existing software as well as software that had to be developed during this research. During and after the build of the simulation tool, every aspect of the program had to be verified and validated. As described in Chapter 7, the program was verified and validated in different steps. Once the program was finished, two different simulated ascent cases were used to determine the validity of the outcomes of the simulation program. These same cases were then used for the comparison between RKF78 and TSI but also to perform a sensitivity analysis on TSI. All these results and the corresponding analyses are described in Chapter 8. From these analyses a number of conclusions could be drawn which are outlined in Chapter 9. During the entire thesis research process, many more questions were formed and many more improvements were envisioned. Unfortunately, not all of these questions could be answered nor all improvements implemented in the scope of this thesis research unfortunately. This is why the same chapter also includes a recommendations section for future research. Hopefully this research is able to reduce the knowledge gap that currently exists concerning the performance of TSI for an ascent trajectory case.

2

PROBLEM BACKGROUND

The majority of this research was performed at NASA's JPL in Pasadena, California. For that reason it was decided that the research of this thesis should focus on something related to either current research being done at JPL or missions being flown by JPL[1]. One of the focuses of JPL these days is Mars and the exploration of the Martian system. There are currently two planned missions to Mars: InSight and Mars 2020. InSight is a mission similar to the Viking and the Phoenix missions and primarily based on the last one. It is a lander that will perform experiments on the surface of Mars at a fixed location. Mars 2020 is the next Mars rover and is based on the current Curiosity rover. It has a similar design but will carry different scientific instruments and will focus more on finding life on Mars. Mars 2020 is part of a larger JPL effort to eventually return samples from the Martian surface back to Earth. This mission concepts has been on the drawing board for several years now, but seems to be getting closer to approval. This sample return mission, or MSR, will consist of three different systems and is spread out over three different missions as shown by the proposed mission architecture in Figure 2.1. Mars 2020 is the first step of this plan where the rover will collect samples from the surface of Mars. These samples will then be put into containment tubes and dropped on the surface of Mars. A second mission will then carry a system that will collect these samples from the surface and deliver them into a Martian orbit (Shotwell *et al.*, 2016). Then a third mission, currently proposed as the Mars 2022 orbiter, will collect the sample containment sphere in orbit and eventually deliver them back to Earth (Woolley *et al.*, 2011). Because of the potential of the MSR mission concept and the amount of research that can be done, the focus was put on this JPL effort when looking at research subjects.

2.1. CHOSEN MISSION

The Mars 2020 rover is an approved mission that is currently in the design phase, and the Mars 2022 is a proposed mission that will have to be build in order to enhance the current communication capabilities between Mars and Earth. Therefore it is almost certain that these two missions will fly. However, this does not guarantee that the entire MSR endeavour will also be realised. This is because there is no official approval yet for the second system that will actually bring the samples from the surface back to earth. However, this does not mean that no research is being done on the return system. Many designs have been envisioned over the past few years and even now engineers cannot decide on the best design. Shotwell *et al.* (2016) shows two of the proposed designs that are currently being considered. One design involves a mobile launch system based on the Curiosity rover, which will fetch the samples and then directly put them into the spherical container. Once this container is completely filled, the sphere will be placed on top of the MAV, which the rover will be carrying on its back, and launch it into Mars orbit. The second design uses two systems, one fetch rover that will grab the samples and return them to the lander, and the lander containing the MAV. Once the lander returns with all the samples, the sample sphere will be put on top of the MAV and launched into orbit. There are advantages and disadvantages to both of these systems and more research is being done to determine the best option. What both of these systems have in common though is that they have to launch the sample sphere into orbit using a MAV. An ascent on another celestial body with an atmosphere has never been attempted before, which makes it a crucial part of the MSR campaign. This is also why in the last several years, many researchers have looked at the MAV launch trajectory problem.

[1] All past, current and proposed JPL missions: http://www.jpl.nasa.gov/missions/ [Accessed 17 November 2016]

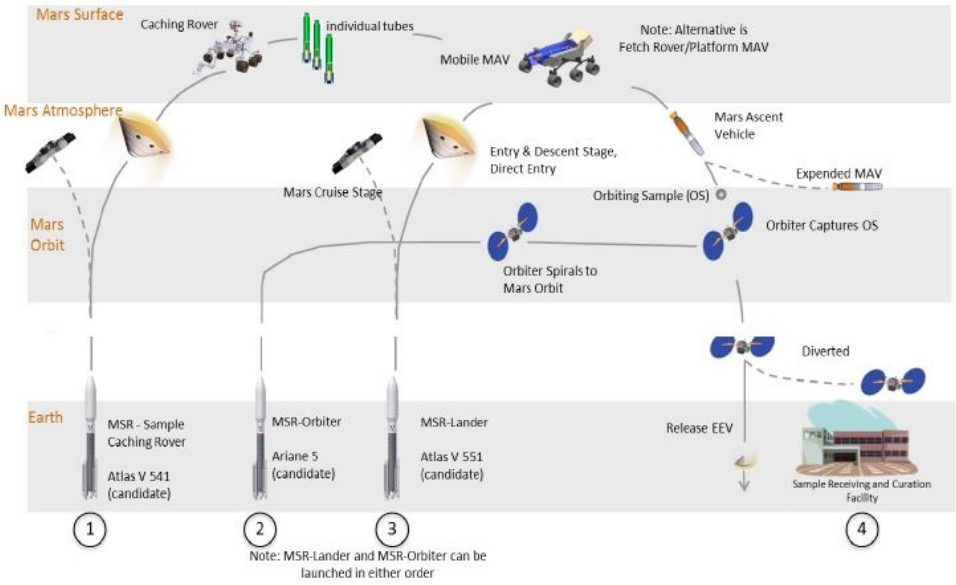


Figure 2.1: Proposed MSR mission architecture (Vaughan *et al.*, 2016).

2.2. PREVIOUS MARS ASCENT RESEARCH

Not only JPL but also many other institutions around the world are working on the Mars ascent problem, for instance the German Aerospace Center (Deutsches Zentrum für Luft- und Raumfahrt) (DLR). A selection of reference research is provided in Table 2.1.

Table 2.1: Previous and current Mars ascent trajectory research.

Author	Organisation	Country
Fanning and Pierson (1996)	Iowa State University	United States
Desai *et al.* (1998)	NASA Langley and JPL	United States
Whitehead (2004, 2005)	Lawrence Livermore National Laboratory	United States
Di Sotto *et al.* (2007)	DEIMOS Engenharia and ESA	Portugal/Europe
Woolley *et al.* (2011)	JPL	United States
Trinidad *et al.* (2012)	Northrop Grumman and JPL	United States
Dumont (2015) (Ongoing)	DLR	Germany
Woolley (2015) (Ongoing)	JPL	United States
Benito and Johnson (2016) (Ongoing)	JPL	United States

Most of the papers before 2015 focus on an older MAV concept, however the newer JPL research focuses more on a Single Stage to Orbit (SSTO) design. This is why this thesis research will also be focusing on the ascent of a SSTO MAV.
Many of the research that was done used either publicly available simulation software or trajectory simulation software that was developed in-house. There are several ways in which an ascent trajectory can be simulated, and these different methods of how to simulate such a launch can results in different outcomes. Therefore, a closer look was taken at the method of ascent trajectory propagation and simulation.

2.3. RESEARCH FOCUS

In order to compute an ascent trajectory, the initial state is taken and provided perturbations the state is propagated until a final condition is met. In most of the research mentioned in Table 2.1 this was done using numerical integration methods. However, often it is not mentioned which specific methods were used. When looking at integration methods, there are a number of methods that are widely used for space related problems, such as the standard integration methods (more information on these standard integration methods can be found in Chapter 4). In recent years however, another method, that has not generally been applied

in space problems, has seen potential to increase performance resulting in faster and more accurate results. This method is called Taylor Series integration (TSI) and was first used in a space trajectory problem by Montenbruck (1992a). A few years later, a comparison between a higher order Runge-Kutta method and TSI was performed on orbital trajectory problems by Scott and Martini (2008) showing that the application of TSI to such problems can be very beneficial. Bergsma and Mooij (2016) showed that TSI shows similar improvements compared to Runge-Kutta-Fehlberg for re-entry cases, where TSI was both faster and more accurate. However, TSI has not yet been applied to ascent cases, leaving a very important knowledge gap that has to be filled.. This is why the Mars ascent trajectory propagation is a good additional test case to add to the knowledge of TSI performance in space related problems. The TSI will have to be compared to a standard integration method to determine its performance.
From the two previous cases (orbital trajectories and re-entry) it is expected that TSI will be faster and more accurate than the standard integration methods. In this thesis, it will be tested whether or not this also holds for ascent cases. However, before this can be tested, an accurate representation of an ascent scenario has to be modelled.

2.4. MAV DESIGN

An important aspect of running an accurate simulation is making sure that the assumed vehicle is representative of the actual vehicle that will be used. Therefore, the most up to date representation of the MAV is required. Over the past 20 years there have been many different iterations of the MAV. Some of the earliest concepts were described by Desai *et al.* (1998); Whitehead (1997); Guernsey (1998); Stone (1999) and were prominently focussed on two-stage liquid rockets (also see Table 2.2. After a change in mission requirements in the early 2000's the leading baseline design changed to a two-stage solid (Whitehead, 2005; Stephenson, 2002; Stephenson and Willenberg, 2006). However, the requirements kept changing and the design kept getting better defined by (among others) Trinidad *et al.* (2012); Sengupta *et al.* (2012); Mungas *et al.* (2012); Mars Program Planning Group (2012) resulting in a two-stage liquid as the updated baseline. The past 4 years however have seen a slight change in the design. Most of the MAV work is now being carried out in-house at JPL. Research is still being performed on the two-stage solids, but SSTO designs are showing that they have an overall better performance. Recently presented designs include a SSTO liquid (Vaughan *et al.*, 2016) and a SSTO hybrid as presented by (Karp *et al.*, 2016). These two papers however focused more on the propulsion systems. Because the mission requirements kept changing leading to many different MAV designs, it can become difficult to keep track of all the differences. Shotwell (2016) provides a good overview of the different designs leading up to the current baseline designs.

Currently there is no clear baseline design, however, the preference leads towards the SSTO designs. Two of those designs are shown in Figure 2.2.

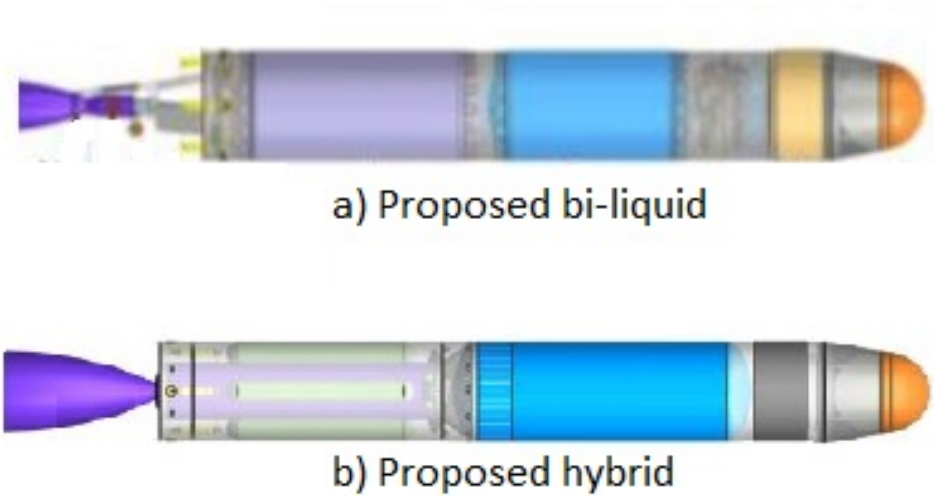


Figure 2.2: Current proposed designs. a) is one of the bi-liquid designs proposed by Vaughan *et al.* (2016). b) is the hybrid design proposed by Karp *et al.* (2016). Not to scale.

Table 2.2: Previous MAV studies.

Author	Rocket	Propellant(s)
Whitehead (1997)	two-stage rockets (comparison study)	solid and liquid (and gel recommended as well)
Guernsey (1998)	two-stage rocket	2x liquid
Desai *et al.* (1998)	two-stage rocket	2x liquid
Stone (1999)	two-stage rocket	hybrid
Stephenson (2002)	two-stage rocket (three different designs)	2x solid (best), solid and liquid or hybrid and 2x gel (best)
Whitehead (2005)	one-, two- and three-stage rockets (variational study)	solid and liquid
Stephenson and Willenberg (2006)	two-stage rocket	2x solid
Sengupta *et al.* (2012)	two-stage rocket	2x liquid
Trinidad *et al.* (2012)	1 to 4 stages (comparison study) two-stage rocket (best)	solid, liquid, hybrid and gel (comparison) 2x liquid (best)
Mungas *et al.* (2012)	single-stage rocket	liquid mono-propellant
Mars Program Planning Group (2012)	undefined rocket	solid and liquid
Vaughan *et al.* (2016)	single-stage rocket	liquid
Karp *et al.* (2016)	single-stage rocket	hybrid

Unfortunately, not much has been published on the specific flight characteristics such as drag coefficient and reference area of these new designs. Fortunately, the size requirements of the MAV have not changed (much) between the 2012 studies and the current designs. Therefore these parameters can be approximated by looking at some of the previous baseline designs such as the two-stage liquid rocket shown in Figure 2.3.

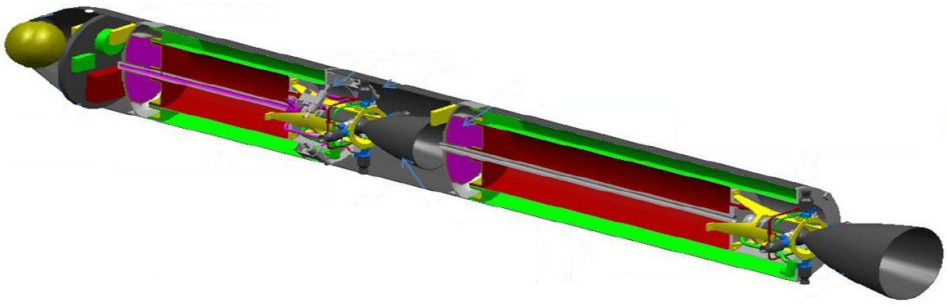


Figure 2.3: Old baseline design (**?**).

This concept had a diameter of approximately 340 mm. This results in a cross sectional area of approximately 0.09079 m^2. Unfortunately, the paper did not mention anything about the drag coefficient. However, **?** provided a general drag coefficient graph as a function of the Mach number for a generalized shape very similar to most MAV concepts. Therefore, this simplified relation, shown in Figure 2.4 is used in this thesis research as well.

These two parameters are used to determine the drag acting on the MAV during the ascent. This drag acts on the MAV in what is called the body frame. One of the problems with trajectories is that different perturbations act differently on the vehicle and are easier to express in different reference frames and systems. In the end however, the state will have to be expressed in one reference frame and system, which means that the drag might have to be transformed to another frame. More information on these different frames is provided in the Section 2.5.

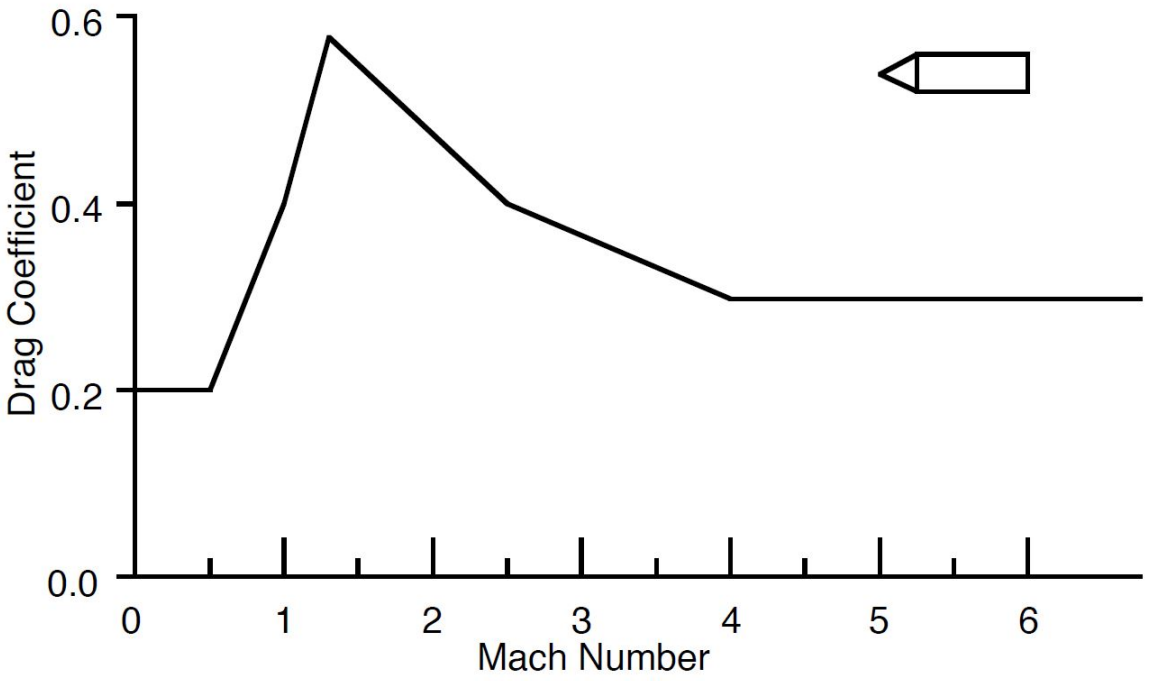


Figure 2.4: Drag coefficient as a function of Mach number Whitehead (2004)

2.5. REFERENCE SYSTEMS

In this research problem, different perturbations are acting in different reference frames. One example is the drag acceleration acting in the body frame. However, the integration will be done in the inertial frame. A collection of all the different frames required is shown in Figure 2.5.

This means that the accelerations have to be translated into the inertial frame through the presented frames. This can be done using so-called transformation matrices ($\mathbb{T}$). These transformation matrices are composed of rotations around different axes over an associated angle. Figure 2.6 shows a general rotation around the x-axis. When this is set in a matrix, the transformation over the x-, y- and z-axis can be defined in the respective transformation matrices as described by Equation (2.1).

$$\mathbb{T}_{\mathbf{x}}(\phi) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\phi & \sin\phi \\ 0 & -\sin\phi & \cos\phi \end{bmatrix}, \mathbb{T}_{\mathbf{y}}(\phi) = \begin{bmatrix} \cos\phi & 0 & -\sin\phi \\ 0 & 1 & 0 \\ \sin\phi & 0 & \cos\phi \end{bmatrix}, \mathbb{T}_{\mathbf{z}}(\phi) = \begin{bmatrix} \cos\phi & \sin\phi & 0 \\ -\sin\phi & \cos\phi & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{2.1}$$

More information on the different reference frames and the transformation matrices can be found in **??**.

Besides different reference frames, there are also different coordinate systems in which the state of a vehicle can be expressed. Three of the most used coordinate systems in space problems are the Cartesian, spherical and Keplerian system, and all of those systems are used in this research.

This also means that the state should be transformed between these different frames. An example of the position expressed both in the Cartesian system as well as the spherical system is shown in Figure 2.7.

From this figure the relation can be described between the two different systems. If the Cartesian coordinates are known, the spherical coordinates can be computed using Equation (2.2).

$$\begin{aligned} r &= \sqrt{x^2 + y^2 + z^2} \\ \delta &= \arcsin\left(\frac{z}{r}\right) \\ \lambda &= \arctan\left(\frac{y}{x}\right) \end{aligned} \tag{2.2}$$

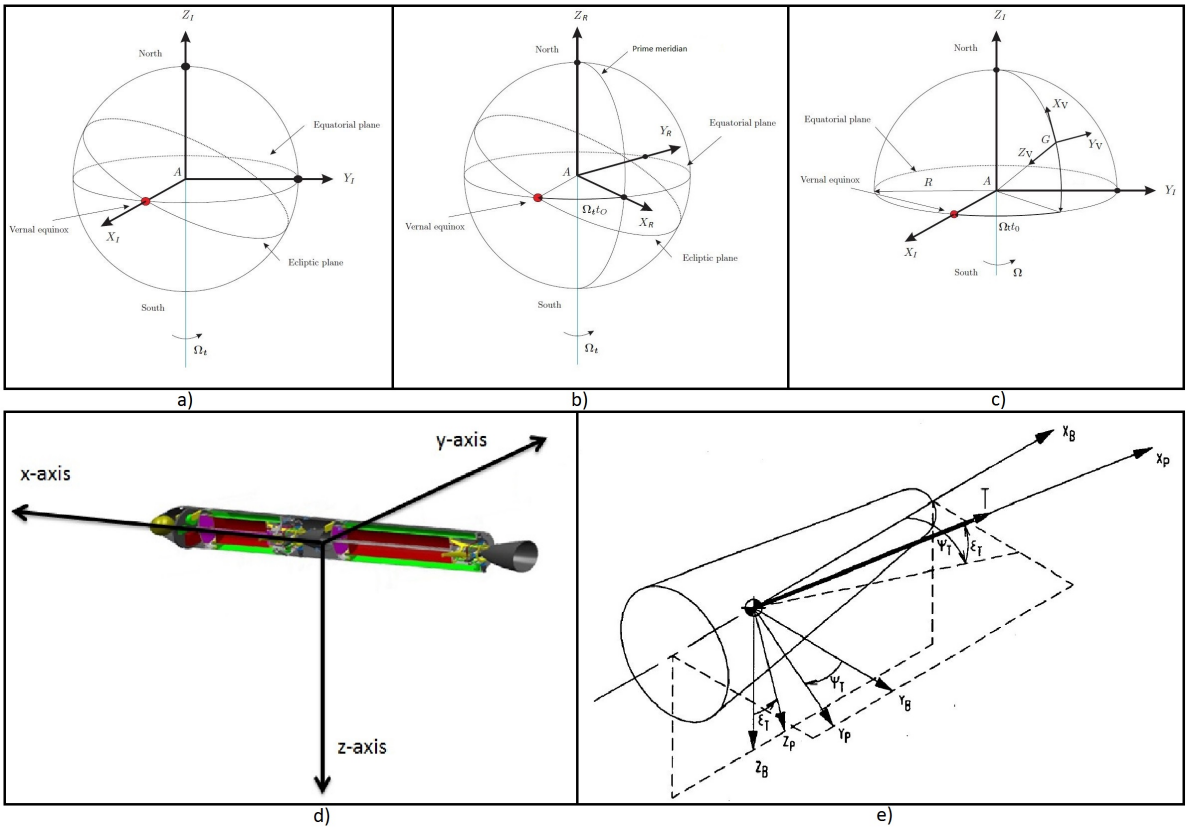


Figure 2.5: a) (Mars centred) Inertial frame, b) (Mars centred) Rotational frame, c) (Vehicle centred) Vertical frame (Mulder *et al.*, 2013). d) (Vehicle centred) Body frame (Trinidad *et al.*, 2012). e) (Vehicle centred) Propulsion frame (Mooij, 1994).

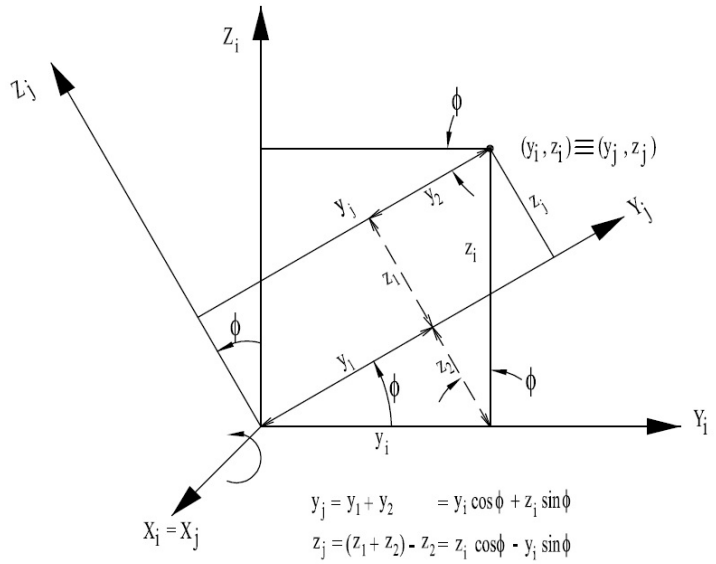


Figure 2.6: General rotation around the x-axis Mooij (2013).

The same calculation can be done from the spherical system to the Cartesian system. The corresponding equations for that calculation are shown in Equation (2.3).

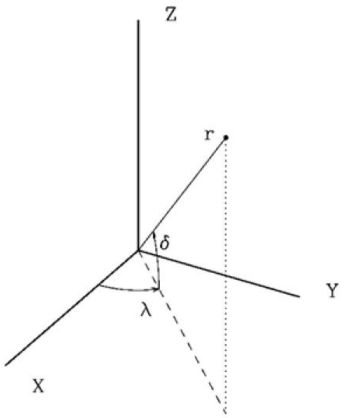


Figure 2.7: Position expressed in the Cartesian and the spherical system Noomen (2013a).

$$\begin{aligned} x &= r\cos\delta\cos\lambda \\ y &= r\cos\delta\sin\lambda \\ z &= r\sin\delta \end{aligned} \tag{2.3}$$

All the coordinate transformations can be found in **??**, which include both position and velocity vectors. These transformations are essential in defining the models and setting up the equations of motion.

3

MODELS

The MAV and the corresponding ascent can be modelled in many different ways. Depending on the desired accuracy of the final solution compared to the real-life expected trajectory, different perturbations and freedom in motion can be defined and included, ranging from a low fidelity to a high fidelity program. The main goal of this thesis is to determine if TSI is an integrator that can be used for ascent cases and determine its performance compared to previously established methods. Therefore, it should be close to a real-life ascent, but not include too many aspects as to make the problem too complex to test. Therefore, it was decided to design a low fidelity program to propagate the trajectory using accelerations in the x-, y- and z-direction (3DOF). The MAV system can then be modelled as a point-mass system with different accelerations acting on it. The change in position can be described by the kinematic equations and the change in velocity by the corresponding dynamic equations. The change in mass is then described by the mass-flow. All these equations are provided in Section 3.1. For this model, three perturbing accelerations will be taken into account: the gravity, the thrust and the drag. The effect of third bodies can be neglected because of the short mission time. All the accelerations are described in Sections 3.2 to 3.4 respectively.

3.1. STATE AND STATE DERIVATIVES

The general model is described in terms of the Cartesian state; position and velocity in the x-, y-, and z-direction and the mass. The notation is shown in Equation (3.1), where m_{MAV} is the mass of the MAV and the subscript I refers to the inertial frame.

$$\mathbf{r} = \begin{pmatrix} x_I \\ y_I \\ z_I \end{pmatrix} \qquad \mathbf{V} = \begin{pmatrix} V_{x_I} \\ V_{y_I} \\ V_{z_I} \end{pmatrix} \qquad m_{MAV} \tag{3.1}$$

In order to determine the next state, the change in the state parameters over time have to be computed. This is done by taking the time derivatives of the state as described by Equation (3.2).

$$\begin{aligned} \dot{x}_I &= V_{x_I} & \ddot{x}_I &= \dot{V}_{x_I} = a_{x_I} \\ \dot{y}_I &= V_{y_I} & \ddot{y}_I &= \dot{V}_{y_I} = a_{y_I} \\ \dot{z}_I &= V_{z_I} & \ddot{z}_I &= \dot{V}_{z_I} = a_{z_I} \end{aligned} \qquad \dot{m}_{MAV} = -\frac{T}{g_0 I_{sp}} \tag{3.2}$$

From this point on, the subscript I is omitted, because the state and the state derivatives are always presented in the inertial frame. If variables have to be presented in any other reference frame the corresponding subscripts will be provided and explained. The accelerations in the x-, y- and z-direction have three contributing components: gravitational acceleration, drag acceleration and thrust acceleration. In this model it is assumed that there is a homogeneous gravitational field acting on the point mass with a direction towards the centre of the inertial frame. The gravitational acceleration can therefore be directly expressed in the inertial frame.
The drag is assumed to work in the opposite direction of the MAV velocity vector, which in turn is assumed

to be in the positive x-direction of the body frame associated with the MAV. This way the MAV velocity, or ground velocity, is acting through the nose of the ascent vehicle. The drag acceleration is therefore expressed in the body frame. This effect in the body frame has to be translated into an acceleration in the inertial frame. This can be done using transformation matrices. A traditional transformation matrix uses an Euler angle to rotate a coordinate frame in a certain reference frame to another. Using the proper Euler, or rotation, angles, the acceleration in the x-, y- and z-direction in the body frame can be transformed into acceleration components acting in the x-, y- and z-direction of the inertial frame. Once it is expressed in the inertial frame it can be added to the effect caused by the gravitational acceleration.
It is also assumed that the nozzle of the MAV engine can be pivoted in two different directions: the y-, and z-direction of the body frame. This means that the thrust acceleration is not acting directly in the body frame, but in what is called the propulsion frame. Because the nozzle can move in two directions, or rather pivot over two different angles, the thrust acceleration can be translated to the body frame using these two thrust angles. Once the thrust acceleration is expressed in the body frame, the same transformations that are used for the drag can be used to determine the thrust acceleration components in the inertial frame.

Combining this in one equation then results in the expression for the acceleration vector as shown by Equation (3.3). The subscript G shows that the parameter is a function of the ground velocity. $\mathbb{T}$ stands for a transformation and the subscript determines the rotation axis. The parameter in the brackets for each transformation is the rotation angle.

$$\begin{pmatrix} a_x \\ a_y \\ a_z \end{pmatrix} = \begin{pmatrix} a_{x,Grav} \\ a_{y,Grav} \\ a_{z,Grav} \end{pmatrix} + \Big|_{\mathbf{I}} \mathbb{T}_{\mathbf{z}}(-\Omega_M t_O + \omega_P) \Big|_{\mathbf{R}} \mathbb{T}_{\mathbf{z}}(-\tau)\, \mathbb{T}_{\mathbf{y}}\left(\frac{\pi}{2} + \delta\right) \Big|_{\mathbf{V}} \mathbb{T}_{\mathbf{z}}(-\chi_G)\, \mathbb{T}_{\mathbf{y}}(-\gamma_G) \Big|_{\mathbf{B}} \left[\begin{pmatrix} a_{Drag} \\ 0 \\ 0 \end{pmatrix} + \dots \right.$$
$$\left. \dots \Big|_{\mathbf{B}} \mathbb{T}_{\mathbf{z}}(-\psi_T)\, \mathbb{T}_{\mathbf{y}}(-\epsilon_T) \Big|_{\mathbf{P}} \begin{pmatrix} a_{Thrust} \\ 0 \\ 0 \end{pmatrix} \right] \tag{3.3}$$

The rotations required to go from the body frame to the inertial frame are all a function of the current state. Both thrust angles are an input value that is set by the user or the program. $-\Omega_M t_O + \omega_P$ represents the angle between the rotating frame and the inertial frame and consists of three parts. Ω_M is the rotational velocity of Mars around it's own axis, t_O is the difference between the current time and the time at which the inertial frame was set on Mars, and ω_P is the angle between the x-axis of the inertial frame and the zero meridian at the time that the inertial frame was set on Mars. In this thesis the inertial frame is set at the beginning of the simulation (so at $t = 0$, $t_O = 0$) and is aligned with the rotating frame (so $\omega_P = 0$). The transformation from the vertical to the rotating frame requires the latitude and longitude of the MAV, which is a function of the current state. Mooij (1994) provides equations to determine the different rotation angles. These expressions can be used to compute the latitude and longitude as well as shown by Equation (3.4).

$$\begin{aligned} r &= \sqrt{x^2 + y^2 + z^2} & x_R &= \cos(\Omega_M t_O)\, x + \sin(\Omega_M t_O)\, y \\ \delta &= \arcsin\left(\frac{z}{r}\right) & y_R &= -\sin(\Omega_M t_O)\, x + \cos(\Omega_M t_O)\, y \\ & & \tau &= \operatorname{atan2}\left(\frac{y_R}{x_R}\right) \end{aligned} \tag{3.4}$$

It can be seen that in order to compute the longitude in the rotating frame (τ) the position of the MAV in the rotating frame was required. This position was found by transforming the inertial position to the rotational position using $\Omega_M t_O$ as described before. Working out that transformation matrix results in the sines and cosines presented in Equation (3.4).

Similarly, the flight-path angle and the azimuth angle can be described as a function of the current state. For this, some intermediate parameters have to be computed first and are given by Equation (3.5).

$$
\begin{aligned}
V_I &= \sqrt{V_x^2 + V_y^2 + V_z^2} \\
V_G &= \sqrt{V_I^2 + \Omega_M^2 \left(x^2 + y^2\right) + 2\Omega_M \left(xy - yx\right)} \\
V_{x,V} &= \left(V_x + \Omega_M y\right) \cdot \left(\sin\left(\delta\right) \sin\left(\Omega_M t_O\right) \sin\left(\tau\right) - \cos\left(\Omega_M t_O\right) \cos\left(\tau\right) \sin\left(\delta\right)\right) + \\
&\quad \left(V_y - \Omega_M x\right) \cdot \left(-\cos\left(\tau\right) \sin\left(\delta\right) \sin\left(\Omega_M t_O\right) - \cos\left(\Omega_M t_O\right) \sin\left(\delta\right) \sin\left(\tau\right)\right) + V_z \cos\left(\delta\right) \\
V_{y,V} &= \left(V_x + \Omega_M y\right) \cdot \left(-\cos\left(\tau\right) \sin\left(\Omega_M t_O\right) - \cos\left(\Omega_M t_O\right) \sin\left(\tau\right)\right) + \\
&\quad \left(V_y - \Omega_M x\right) \cdot \left(\cos\left(\Omega_M t_O\right) \cos\left(\tau\right) - \sin\left(\Omega_M t_O\right) \sin\left(\tau\right)\right) \\
V_{z,V} &= \left(V_x + \Omega_M y\right) \cdot \left(\cos\left(\delta\right) \sin\left(\Omega_M t_O\right) \sin\left(\tau\right) - \cos\left(\Omega_M t_O\right) \cos\left(\tau\right) \cos\left(\delta\right)\right) + \\
&\quad \left(V_y - \Omega_M x\right) \cdot \left(-\cos\left(\tau\right) \cos\left(\delta\right) \sin\left(\Omega_M t_O\right) - \cos\left(\Omega_M t_O\right) \cos\left(\delta\right) \sin\left(\tau\right)\right) - V_z \sin\left(\delta\right)
\end{aligned}
\tag{3.5}
$$

Then the angles can be computed using those parameters as shown by Equation (3.6).

$$
\begin{aligned}
\chi_G &= \operatorname{atan2}\left(\frac{V_{y,V}}{V_{x,V}}\right) \\
\gamma_G &= -\arcsin\left(\frac{V_{z,V}}{V_G}\right)
\end{aligned}
\tag{3.6}
$$

In this case, a transformation from the inertial frame to the vertical frame was required to get information on the velocity in the vertical frame. This transformation is written out at sines and cosines in Equation (3.5) resulting in lengthy expressions.

Now that the rotation angles from the body frame to the inertial frame are known, only the thrust angles and the different accelerations have yet to be defined. This is done in the next sections.

3.2. GRAVITY

The gravity acceleration is a function of the position of the MAV and the standard gravitational parameter of the planet, in this case Mars. Often, irregularities in the mass distribution of a planet also have to be taken into account ($J_{2,0}$ etc.), however, because the mission time is very short and the MAV does not even complete one revolution around Mars, these effects can be neglected. Therefore, the gravitational acceleration in the inertial frame can be represented by Equation (3.7).

$$
\begin{pmatrix} a_{x,Grav} \\ a_{y,Grav} \\ a_{z,Grav} \end{pmatrix} = \begin{pmatrix} -\mu_M \dfrac{x}{r^3} \\ -\mu_M \dfrac{y}{r^3} \\ -\mu_M \dfrac{z}{r^3} \end{pmatrix}
\tag{3.7}
$$

3.3. THRUST

The thrust acceleration in the propulsion frame is simply the thrust divided by the current mass of the MAV, which can be written as shown by Equation (3.8).

$$
a_{Thrust} = \frac{T}{m_{MAV}}
\tag{3.8}
$$

However, this acceleration has to be transformed to the body frame using the thrust angles. These thrust angles are the thrust elevation (ϵ_T) and thrust azimuth (ψ_T) angle, and are defined in Figure 3.1.

These angles will not be computed, but are set as an input for the program.

3.4. DRAG

The drag acceleration can be represented similarly to the thrust acceleration taking into account that the drag is acting in the negative x-direction in the body frame. The acceleration can then be written as Equation (3.9).

$$
a_{drag} = -\frac{D}{m_{MAV}}
\tag{3.9}
$$

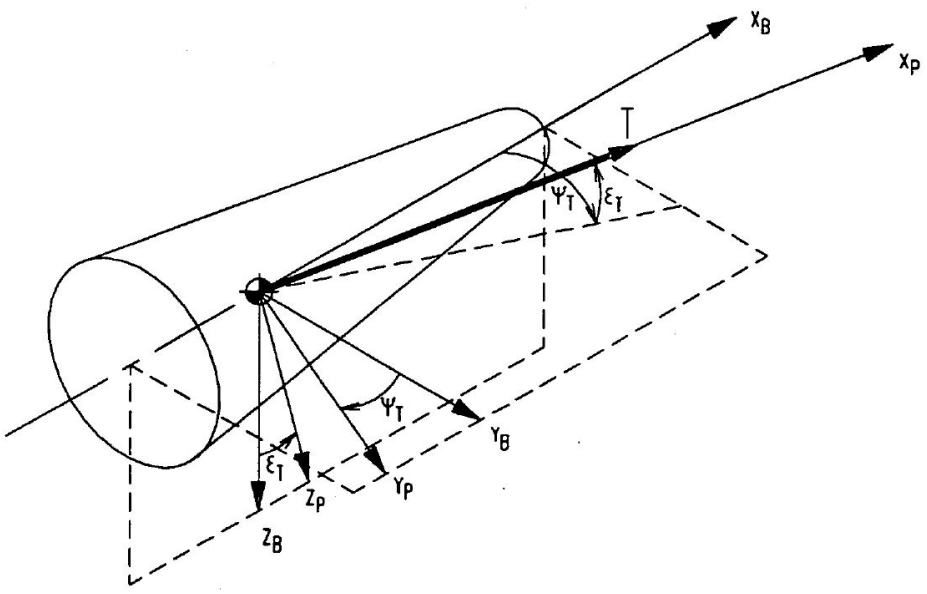


Figure 3.1: Definition of the thrust elevation and thrust azimuth angle with respect to the body frame (Mooij, 1994).

Here the drag is given by Equation (3.10).

$$D = \frac{1}{2}\rho V_G^2 S C_D \tag{3.10}$$

Where S is the reference area of the MAV set by the user, ρ is the air density which is a function of the altitude $h\,(= r - R_{MOLA})$ and C_D is the drag coefficient of the MAV which is a function of the Mach number. The Mach number in turn is a function of the speed of sound, which depends on the air temperature as can be seen in Equation (3.11).

$$\begin{aligned} M &= \frac{V_G}{a} \\ a &= \sqrt{\gamma_a R_a^* T_a} \quad \text{where} \quad R_a^* = \frac{R_a}{M_a} \end{aligned} \tag{3.11}$$

Here, R_a is the molar gas constant, γ_a is the adiabetic index for Mars, which according to Ho *et al.* (2002) can be set as ~1.35. M_a is the molecular mass of the Martian atmosphere and depends on it's composition. Williams (2015) mentions a volumetric composition of Mars which results in a mean molecular mass of 0.04334 kg/mol. This value has been used in this research to compute the speed of sound.

The air pressure and temperature (T_a) change as a function of the altitude of the MAV. This means that a pressure and temperature model of the atmosphere of Mars will have to be used to approximate these values as the ascent vehicle moves through the atmosphere. This is done through a function fit of an atmospheric model, as is described in Section 3.4.1.

The drag coefficient changes with the Mach number, however, this change is not constant over the entire Mach range. Therefore, this also requires a function fit in order to be used in the simulation program. This fit is presented in Section 3.4.2.

3.4.1. ATMOSPHERIC GRAPH 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 Mars Orbiter Laser Altimeter (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 3.2 and 3.3 respectively for 9 latitude and longitude combinations including the launch site itself.

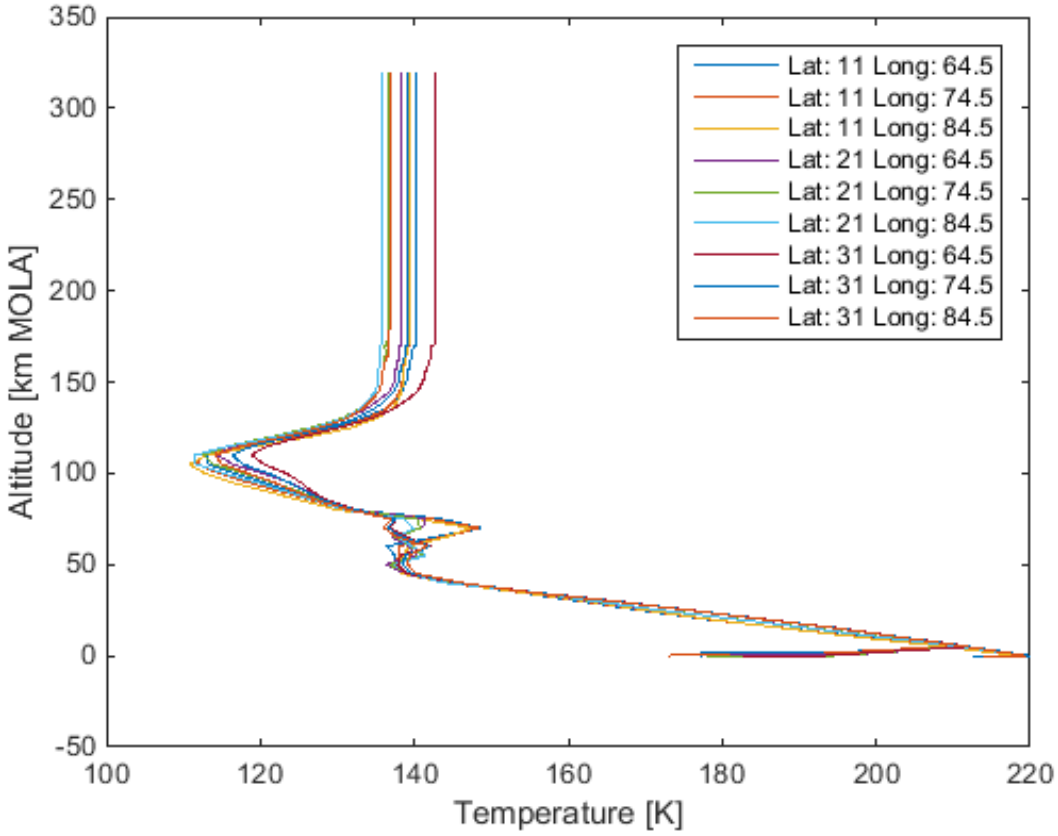


Figure 3.2: Temperature data generated with Mars-GRAM 2005 showing 9 different latitude and longitude combinations

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 function 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 3.4. 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. More information on the fitting process and early results is provided in Appendix B.

Each section was fit with a polynomial function of the n^{th} order where the function is represented by Equation (3.12). The last section shows a constant temperature, thus the temperature of the launch site curve was chosen to represent this final section, which is equal to 136.5 K.

$$y = p_n x^n + p_{n-1} x^{n-1} + \cdots + p_1 x + p_0 \tag{3.12}$$

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 3.1 shows the orders that were required and the deviations to the launch site temperature-altitude curve. The actual corresponding parameters are provided in Table 3.2. 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

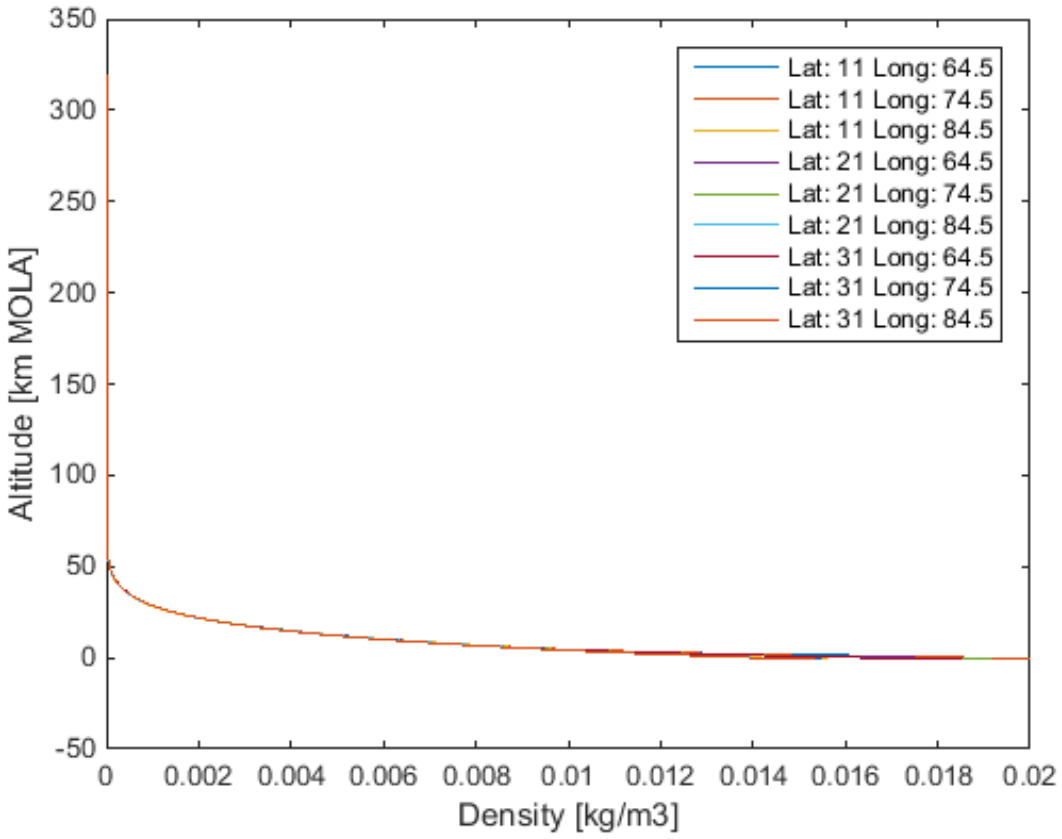


Figure 3.3: Density data generated with Mars-GRAM 2005 showing 9 different latitude and longitude combinations

and were thus treated as outliers.

Table 3.1: Temperature curve fit data all with respect to the launch site curve (Latitude and longitude of the launch site)

Section	**Altitude range [km MOLA]**	**Order**	**Maximum polynomial standard deviation [K]**	**Maximum polynomial difference [K]**	**Maximum data curves difference [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 3.2: Temperature curve fit parameters (rounded to 3 decimal points)

Section	p_8	p_7	p_6	p_5	p_4	p_3	p_2	p_1	p_0
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	$-1.523 \cdot 10^{4}$	$2.236 \cdot 10^{5}$

The complete polynomial fit for the launch site curve for the temperature is shown in Figure 3.5.

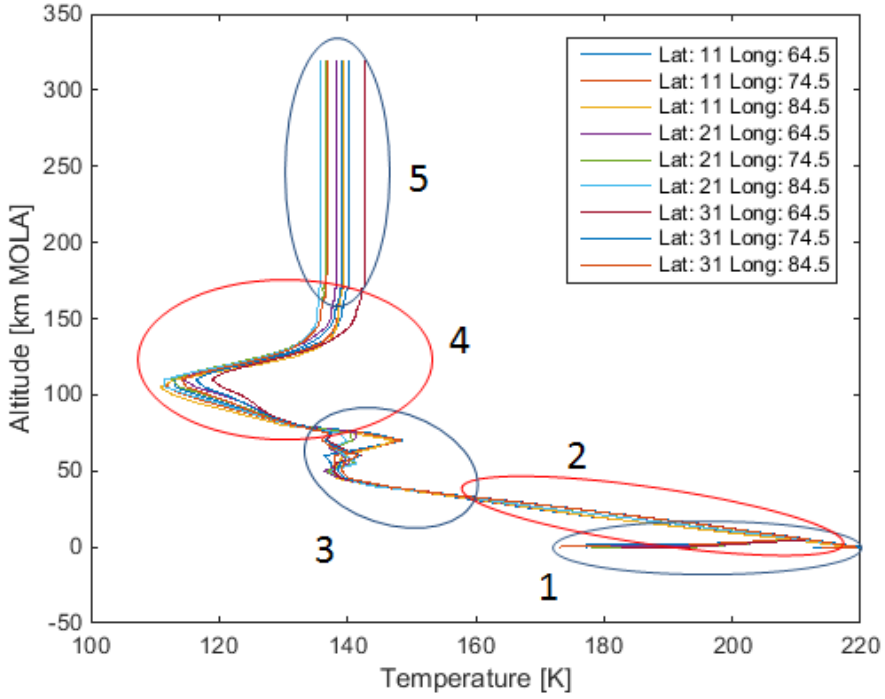


Figure 3.4: Different temperature curve sections

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 3.6.

With the data represented in the logarithmic domain, again a polynomial function can be fit. The total fit would then satisfy Equation (3.13).

$$y = exp\left(p_n x^n + p_{n-1} x^{n-1} + \cdots + p_1 x + p_0\right) \tag{3.13}$$

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 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 3.3 and 3.4 and the polynomial and exponential fit curves are shown in Figures 3.7 and 3.8 respectively.

3.4.2. DRAG COEFFICIENT GRAPH FUNCTION FIT

Similar to the temperature and density curves, the relation between Mach number and drag coefficient, as depicted in Figure 2.4, 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.

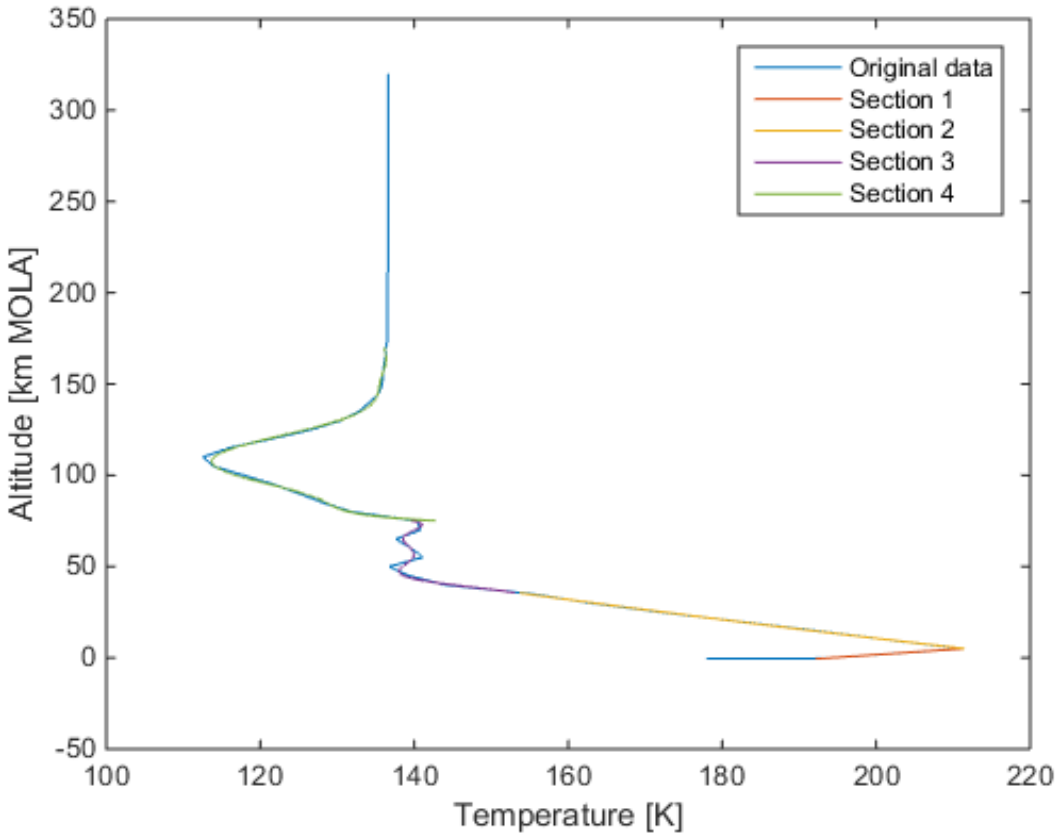


Figure 3.5: All section fits for the launch site temperature data curve

Table 3.3: Density curve fit data (10^{th} order polynomial) with respect to the launch site curve (Latitude and longitude of the launch site)

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

Table 3.4: Density curve fit parameters (rounded to 3 decimal points)

p_{10}	p_9	p_8	p_7	p_6	p_5	p_4	p_3	p_2	p_1	p_0
$2.287 \cdot 10^{-21}$	$-3.724 \cdot 10^{-18}$	$2.559 \cdot 10^{-15}$	$-9.620 \cdot 10^{-13}$	$2.146 \cdot 10^{-10}$	$-2.884 \cdot 10^{-8}$	$2.273 \cdot 10^{-6}$	$-9.604 \cdot 10^{-5}$	$1.414 \cdot 10^{-3}$	-0.0962	-4.172

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 (C_D is 0.2 and 0.3 respectively). 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 3.5 and the curve fit is shown in Figure 3.9.

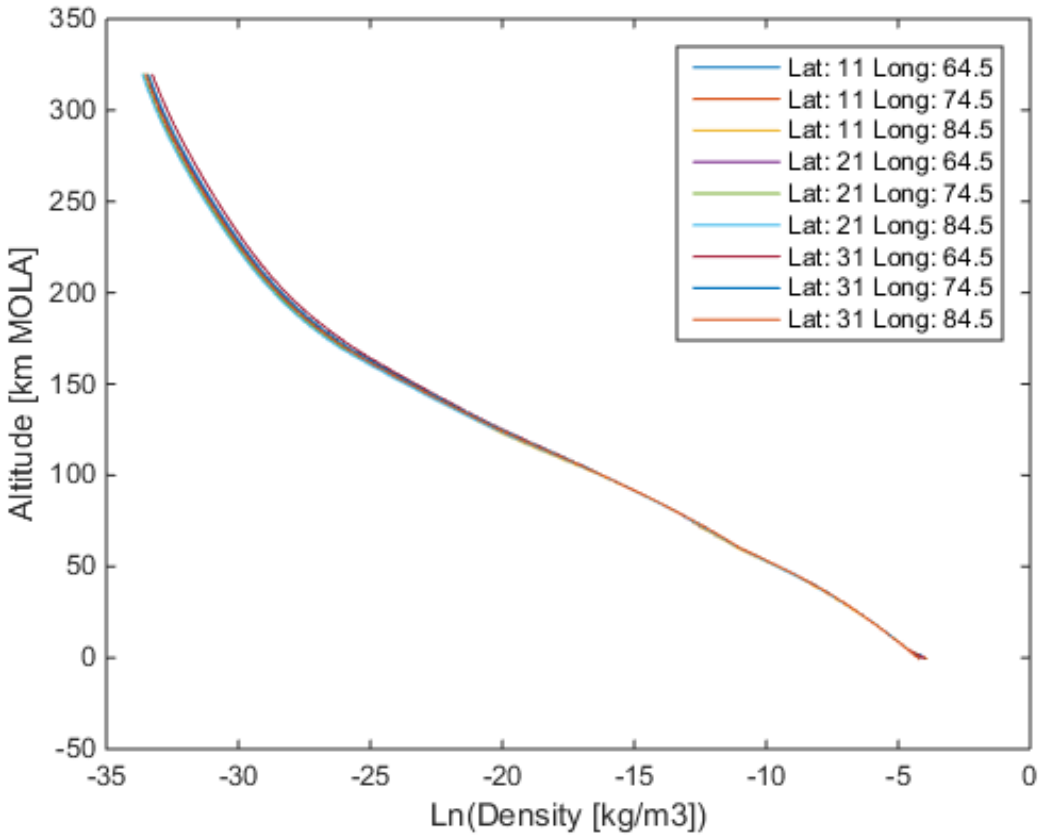


Figure 3.6: Natural logarithmic plot of the density data

Table 3.5: Drag coefficient curve fit parameters (rounded to 3 decimal points)

Section	**Mach range**	$\mathbf{p_1}$	$\mathbf{p_0}$
2	0.5 to 1	0.400	$-2.483 \cdot 10^{-16}$
3	1 to 1.3	0.567	-0.167
4	1.3 to 2.5	-0.142	0.754
5	2.5 to 4	-0.0667	0.567

3.5. CIRCULARISATION

After the integration has been completed, a circularisation burn will take place to estimate the required propellant mass to arrive in the desired orbit. This will include the ΔV required to change the flight-path angle, change the inclination and adjust the orbital velocity. A simplified diagram of such a procedure is presented in Figure 3.10. The equations used to perform this instantaneous burn are based on the equations presented by Wakker (2010).

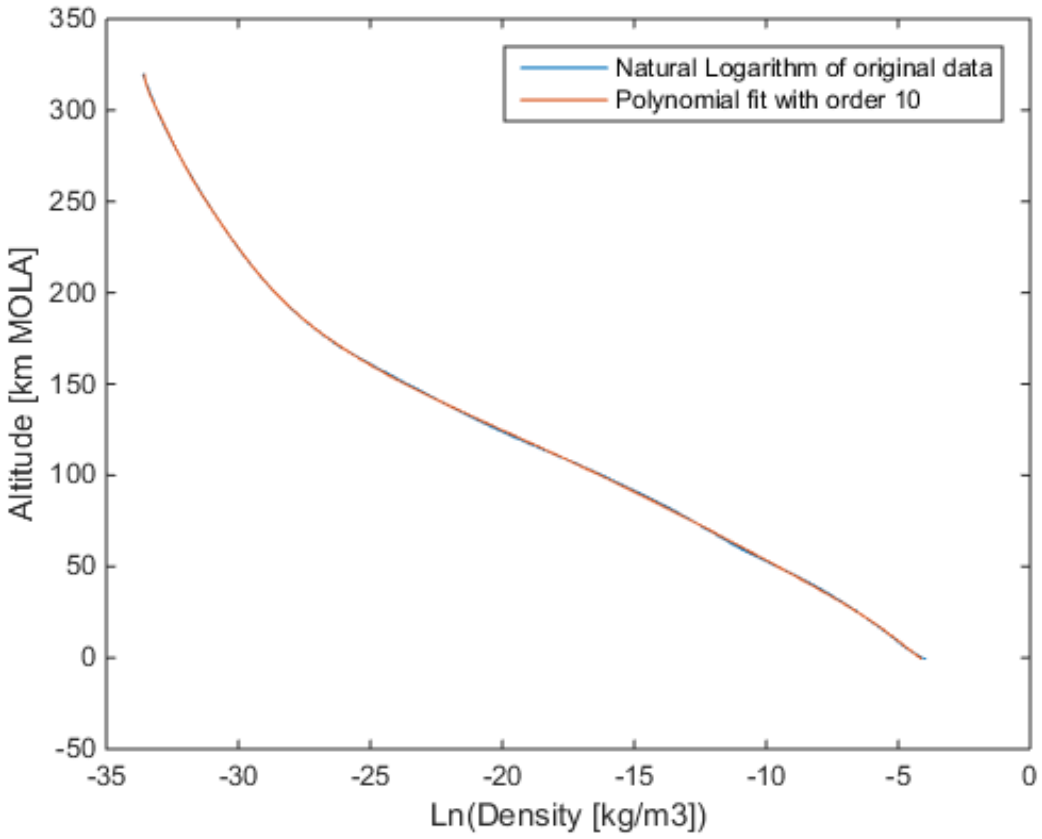


Figure 3.7: Polynomial fit for the launch site density data curve

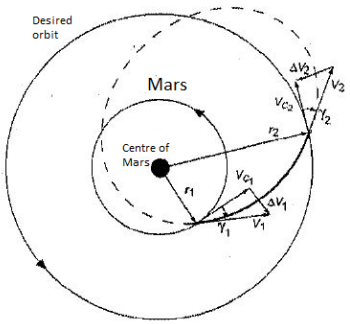


Figure 3.10: Circularisation into the desired orbit (**?**).

The first step is to convert the final Cartesian state obtained at the end of the integration to Kepler elements. This conversion is a standard conversion that can be found in Wakker (2010) and requires the standard gravitational parameter of Mars (μ_M), which is taken to be 4.2828314 km^3/s^2. Then using the vis-viva equation presented in Equation (3.14) the current orbital velocity of the launch trajectory can be computed. Here the subscript co stands for current orbit and a is the semi-major axis of the launch trajectory.

$$V_{co} = \sqrt{\mu_M \left(\frac{2}{r_{co}} - \frac{1}{a_{co}} \right)} \tag{3.14}$$

Then he required orbital velocity is a simplified representation of the vis-viva equations because it is assumed that the required orbit is always circular, see Equation (3.15). Here do stands for desired orbit.

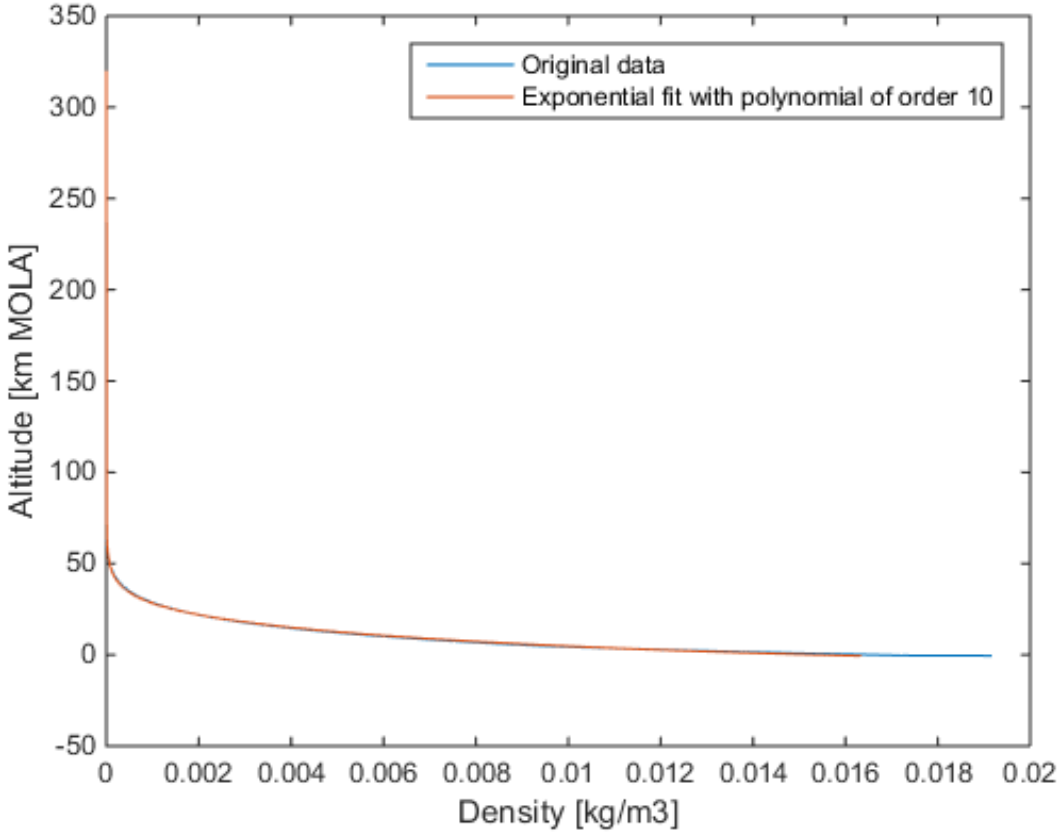


Figure 3.8: Exponential fit for the launch site density data curve

$$V_{do} = \sqrt{\frac{\mu_M}{R_M + r_{do}}} \tag{3.15}$$

Because a change in flight-path angle might be required, the current flight-path angle has to be included into the change in velocity as well. For this, the semi-latus rectum (p) of the current orbit has to be computed first and then the cosine of the flight-path angle can be computed as shown by Equation (3.16).

$$\begin{aligned} p_{co} &= a_{co}\left(1 - e_{co}^2\right) \\ \cos\gamma_{co} &= \frac{\left(\frac{p_{co}}{r_{co}}\right)}{\left(\sqrt{2\left(\frac{p_{co}}{r_{co}}\right) - \left(1 - e^2\right)}\right)} \end{aligned} \tag{3.16}$$

The required ΔV needed to reach the desired orbit can now be computed using Equation (3.17).

$$\Delta V = \sqrt{V_{do}^2 + V_{co} - 2V_{do}V_{co}\cos\gamma_{co}\cos\left(i_{do} - i_{co}\right)} \tag{3.17}$$

Finally, the required propellant mass can be computed using Tsiolkovsky's equation as shown by Equation (3.18).

$$m_{prop} = m_{MAV}\left(1 - e^{\frac{-\Delta V}{I_{sp}g_0}}\right) \tag{3.18}$$

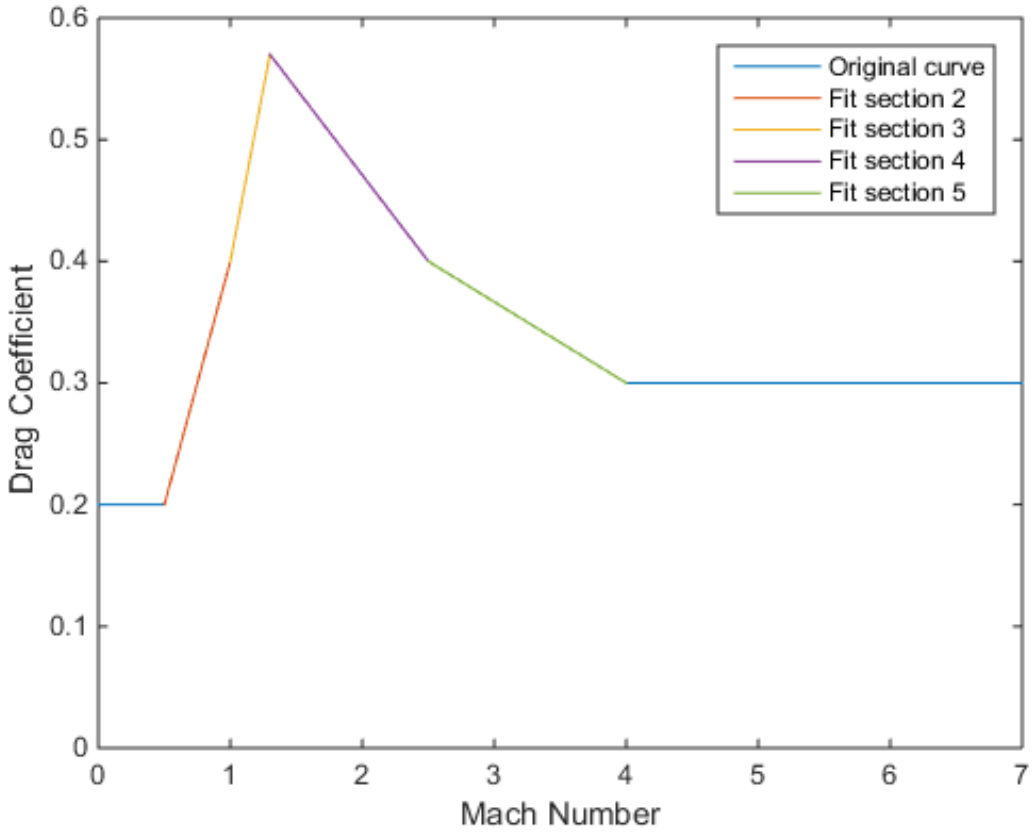


Figure 3.9: All section fits for the drag coefficient - Mach curve

4

STANDARD INTEGRATION METHODS

Propagation refers to the process of modelling/predicting the manner in which an object (for instance) will progress (in time) from a starting point, usually given an initial condition. Such an initial condition could be a constant force or another kind of perturbation. In orbital computations, numerical integration is often used to model this behaviour, because of the irregularities in the dynamic environment (many perturbations) and the absence of analytical solutions. The solution then provides an estimate of the trajectory and the state of the Spacecraft (s/c)(Hofsteenge, 2013). There are several different types of numerical integrators that all try to predict the s/c behaviour in different ways. These types are described in Section 4.1. In Section 4.2 the method is chosen that will be used in this research to compare the performance of the TSI method with.
Numerical integration methods do have one major drawback compared to analytic methods, and that is the accuracy of the produced result. Numerical methods come close to the actual result but usually do not reach it. Therefore, there will always be a certain error associated with the answer. One of the most important errors is called the local (and total) truncation error, which is caused by the numerical inaccuracy of the method itself, and are usually the largest errors. A second error can be introduced by the round-off properties of the used computer, which then also depends on the number of evaluations required for each of the integration methods (more evaluations means more round-off errors). These round-off errors are due to the fact that computers can only accurately represent a number up until a certain number of decimal figures. According to Milani and Nobili (1987) there are also specific errors that arise when integrating space-related problems. Instability errors can occur if the simulated system is chaotic or if the step-size is too large.
Another, non-method specific and not necessarily integration related, error source is the mistakes made in the physical model representing the problem in the simulation. Such errors can occur when forces and disturbances are not (properly) taken into account in the assumptions made to create the physical model.
Fortunately, many of the standard integration methods already contain functions that approximate these errors and then include those approximations to come up with a better solution, or try to minimize these errors by using multiple approximations. The focus of this chapter will thus be on the different integration methods and not the different methods of determining the different errors.

4.1. DIFFERENT INTEGRATOR TYPES

A number of different numerical integration methods exist, however in this section they will be split based on how they work. In this case they have been split into single-step (Section 4.1.1) and multi-step methods (Section 4.1.2) based on Noomen (2013b). The methods can also be split based on the used step-size; either a fixed step-size that does not change during the integration, or a variable step-size that changes per step (or sometimes even during the step). Finally, the methods can also be categorised by either being explicit or implicit. An explicit method uses the information provided at the start of the integration (the current state $\mathbf{x}_i$ and sometimes previous states) to determine the next state $\mathbf{x}_{i+1}$. Whereas an implicit method uses the same information to determine $\mathbf{x}_{i+1}$, but once this first solution is obtained, it uses this next state as an approximation of the solution and feeds it back into the method in order to determine a more accurate solution. This requires iteration.
The numerical approximation of the next state can be written as the current state plus the change during one time-step. This change is defined as the step-size h times the increment function Φ as shown by Equa-

tion (4.1), which changes depending on the method used. Here, η represents the total numerical approximation.

$$\mathbf{x}(t_0 + h) \approx \mathbf{x}_0 + h\mathbf{\Phi} = \eta(t_0 + h) \tag{4.1}$$

4.1.1. SINGLE-STEP

Single-step methods solely use the information at the current point to determine the approximation of the next state. This means tht the information of the previous points is neglected and usually not saved (Noomen, 2013b). Examples of the most simple explicit, fixed step-size, single-step methods are Euler, Mid-point and Runge-Kutta 4^{th} order (RK4) (Hofsteenge, 2013). Euler uses the information at the current state to directly compute the estimate of the next state. Mid-point already incorporates an extra estimation where it first computes a point at half the step-size and then uses that information to approximate the next state, and RK4 takes 4 points into account and takes the weighted average of those to come up with a solution (the starting point, two mid-points and a final point). Many methods exists that are based on these simple methods, for instance the Mid-point method based high-order extrapolation (a.k.a. DIFEX2) (explicit) method (Deuflhard *et al.*, 1994).
Many more methods however are based on the RK4 principle such as Runge-Kutta-Nyström (RKN, with variations such as RKN7(6)9) (implicit) (Montenbruck, 1992b; Dormand *et al.*, 1987), Runge-Kutta-Nyström 12^{th} order (RKN12) (implicit) (Montenbruck, 1992b) and Runge-Kutta-Fehlberg 4^{th} $\left(5^{th}\right)$ order (RKF45) and Runge-Kutta-Fehlberg 7^{th} $\left(8^{th}\right)$ order (RKF78) (Fehlberg, 1969, 1968). These last two integrators are slightly different since they are still explicit, but use a variable step-size.

4.1.2. MULTI-STEP

Compared to the single-step method, the multi-step method does use the information of the previous points to estimate the next state. Usually reaching as far back as the previous three points such as the Adams-Bashforth 4^{th} order (AB4) method (Noomen, 2013b), this being the only difference between this method and the RK4 method. Extending this method creates the explicit Adams-Bashforth 6^{th} order (AB6) method. An example of an implicit, fixed step-size, multi-step method is the Adams-Moulton method. It uses a polynomial to interpolate the function values in order to approximate the next state (Noomen, 2013b). Explicit and implicit methods can also be combined to create so-called Predictor-Corrector methods. Here an initial guess is created by the explicit method, which is then fed into the implicit part in order to correct and improve the estimate. Examples of Predictor-Corrector methods are Adams-Bashforth-Moulton 4^{th} order (ABM4) and Adams-Bashforth-Moulton 12^{th} order (ABM12) (Noomen, 2013b; Montenbruck, 1992b).
All previously mentioned multi-step methods use a fixed step-size. Variable step-size methods also exist, such as Shampine-Gordon (SG) (explicit) (Berry, 2004; Meijaard, 1991) and Störmer-Cowell 14^{th} order (SC14) (Predictor-Corrector) (Berry, 2004; Ramos and Vigo-Aguiar, 2005).

4.2. CHOSEN METHOD FOR COMPARISON

In order to choose the method that will be used to compare the performance of TSI to, it was important to understand which methods were readily available. If a method is already validated and available for use a lot of time can be saved. Therefore, an inquiry of available methods is made in Section 4.2.1. Another important criteria is that the performance of TSI should be able to be compared to previous study cases. The easiest way of doing that is to see what methods other researchers used to compare TSI to (see Section 4.2.2. Using the information from Sections 4.2.1 and 4.2.2 a decision could be made on the final comparison method as is described in Section 4.2.3.

4.2.1. TUDAT METHODS

One of the software tools used in this thesis is Tudat (see Section 6.1.1 for more information). It is a computational toolbox that already comes with a number of useful functions. It also contains a number of pre-programmed validated numerical integrators. A list of the available integration methods is provided in Table 4.1.

Most of these methods are very similar except RK4 which is a fixed step-size method, and Dormand-Prince 8^{th} $\left(7^{th}\right)$ order (DOPRIN87) which is actually a Runge-Kutta method similar to RKF with an accuracy order 8(7) using the formulations as described by Prince and Dormand (1981) according to Weeks and Thrasher

Table 4.1: Available Tudat integrators (Dirkx *et al.*, 2016)

Method	Kind of method	File	Ready for use
RK4	explicit, fixed step-size, single-step	rungeKutta4Integrator.h	Yes
RKF45	explicit, variable step-size, single-step	numericalIntegrator.h & rungeKuttaVariableStep-SizeIntegrator.h & rungeKuttaCoefficients.h/.cpp	Yes
RKF56	explicit, variable step-size, single-step	numericalIntegrator.h & rungeKuttaVariableStep-SizeIntegrator.h & rungeKuttaCoefficients.h/.cpp	No
RKF78	explicit, variable step-size, single-step	numericalIntegrator.h & rungeKuttaVariableStep-SizeIntegrator.h & rungeKuttaCoefficients.h/.cpp	Yes
DOPRIN87	explicit, variable step-size, single-step	numericalIntegrator.h & rungeKuttaVariableStep-SizeIntegrator.h & rungeKuttaCoefficients.h/.cpp	Yes

(2007). At the time of the research, the Runge-Kutta-Fehlberg 5^{th} (6^{th}) order (RKF56) method that was pre-programmed into Tudat was unfortunately not available for use.

4.2.2. METHODS USED IN PREVIOUS RESEARCH

Out of the research done on integration methods for space missions, the most relevant is the research that was done on TSI. The research performed by Scott and Martini (2008) focused on orbital trajectories and used RKF8(9) and the researched performed by Bergsma and Mooij (2016) focussed on re-entry cases and used RKF56. Unfortunately, RKF8(9) was not available through Tudat and RKF56 was out of commission at the time of this thesis research. However, it is good to notice that both researchers used higher order RKF methods, which encourages the use of similar methods for this research and fortunately similar methods are available.

4.2.3. CHOSEN METHOD

Considering the available methods and the methods used in previous TSI research it was determined that the RKF methods and DOPRIN87 would be tested in the early phases of the verification process. In this case RK4 would serve as a back-up. During the verification of the standard integrator it was found that RKF78 showed the best performance when dealing with these ascent problems, which is why it was chosen as the integration method to which TSI was later compared.

4.3. WORKINGS OF RKF

The principle behind RKF is perhaps best explained by first looking at a simpler Runge-Kutta method such as RK4. Noomen (2013b) provides a simple explanation of the workings of RK4 that have been adapted here. In this case, use is made of the general formulation for the numerical approximation as was shown in Equation (4.1). The increment function for RK4 is then presented by Equation (4.2). In this case four points are used to approximate the next state as visualised in Figure 4.1.

$$\Phi_{RK4} = \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4) \tag{4.2}$$

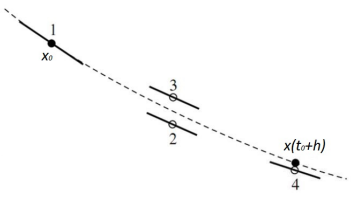


Figure 4.1: Principle of RK4 for a single parameter Noomen (2013b).

Runge-Kutta works with the time-derivatives at those points. The derivative at the first point is called k_1, at the second point k_2 etcetera. These time derivatives are used both in the increment function but also in

the process of determining the derivative of the next intermediate point as shown by Equation (4.3).

$$\begin{aligned} k_1 &= f'(t_0, \mathbf{x}_0) \\ k_2 &= f'(t_0 + h/2, \mathbf{x}_0 + hk_1/2) \\ k_3 &= f'(t_0 + h/2, \mathbf{x}_0 + hk_2/2) \\ k_4 &= f'(t_0 + h, \mathbf{x}_0 + hk_3) \end{aligned} \tag{4.3}$$

Equations (4.2) and (4.3) can also be generalized as presented by Equation (4.4). In this case b_i represents the different weights for each of the points with the corresponding step-size weights c_i, n is the order and $a_{i,j}$ are the coefficients of the previous time-derivatives.

$$\begin{aligned} \Phi_{RK} &= \sum_{i=1}^{n} b_i k_i \\ \text{with} \quad k_i &= f'\left(t + c_n h, y + h \sum_{j=1}^{n-1} a_{i,j} k_j\right) \end{aligned} \tag{4.4}$$

This general formulation can now also be used to compute the higher order Runge-Kutta approximations given that $a_{i,j}$, b_i and c_i are provided. These sets of numbers can be found in various literature, and different people have come up with different sets. Fehlberg was one of these people. In order to get a more accurate solution he decided to use two different methods. In the case of this research, RKF78 was used, which means that a 7^{th} order and 8^{th} order solution can be computed using Equation (4.4). The 7^{th} order solution is then used as the method solution and the difference between the 8^{th} and the 7^{th} order solution results in an error estimate which can be used to determine the next step-size. This step-size is thus computed such as to minimize this error estimate. This means that the only difference between RKF78 and RKF45 (for instance) is the coefficient set (a, b and c's).

5

TAYLOR SERIES INTEGRATION

Taylor Series integration (TSI) is different from the previously mentioned integrators, because it only uses the information at the current point to predict the next point. However, it does this using higher order derivatives at that point. These higher order derivatives have to be obtained as a function of the current state and the first order state derivatives. As soon as the K^{th} order derivative as been found, a Taylor Series can be set up to determine the next state as a function of the chosen step-size. This is all explained and described in more detail in Section 5.1. In this thesis, TSI is the main focus of the analysis. The performance with respect to the RKF integrators is tested. In order to be able to do this for the given model, a number of equation sets can be identified which have to be written in such a format that it can be used by TSI. These are described, and where needed derived, in Section 5.2.

5.1. GENERAL THEORY

TSI has been used to solve ordinary differential equations since the early 1960s (Scott and Martini, 2008). However, the first modern implementation of TSI in a space trajectory problem was provided by Montenbruck (1992a). Scott and Martini (2008) were able to implement this TSI method into the SNAP trajectory propagator which is the implementation that is used in this thesis. The method is described in Section 5.1.1 and the used step-size method is discussed in Section 5.1.2. This step-size sometimes has to be reduced if a new section in the temperature or the drag coefficient model is reached. A root finding method is described in Section 5.1.3 that can be used to determine at what step-size the next section is reached.

5.1.1. WORKINGS OF TSI

The formulation used in this thesis is based on the one used by Scott and Martini (2008).

TSI uses the current state and its derivatives to determine the next state by computing its Taylor Series described in Equation (5.1). The state vector is represented by $\mathbf{X}$ with the corresponding vector for the initial conditions $\mathbf{X}_0$. Each of the variables can at any point be represented by $x_n(t)$. For a given step-size h and an order $K \in \mathbb{R}$ (chosen by the user) the updated state $x_n(t+h)$ can be represented by the Taylor Series with $n = 1,\ldots,7$ in this case (7 variables: three position, three velocity and one mass). The exact solution is obtained by adding $T_{n,K}$, which is the truncation error for the n^{th} variable using K terms.

$$x_n(t+h) = \sum_{k=0}^{K} \frac{x_n^{(k)}(t)}{k!} h^k \quad + \quad T_{n,K} \tag{5.1}$$

This particular TSI method uses recurrence relations to determine the k^{th} order derivatives and only requires the current state (x_n) and its first derivatives ($x_n' = u_n$). Each recurrence relation describes a certain operation in the state derivative, such as: multiplication, division, power, exponential, sines and cosines. Equation (5.2) shows the recurrence relations for multiplication ($w(t) = f(t)\,g(t)$) and division ($w(t) = \frac{f(t)}{g(t)}$) respectively (the other operations are described later in this chapter when they are required).

$$\begin{aligned}
&\text{For multiplications} \quad W(k) = \sum_{j=0}^{k} F(j)\,G(k-j) \\
&\text{For divisions} \quad W(k) = \frac{1}{g(t_0)}\left[F(k) - \sum_{j=1}^{k} G(j)\,W(k-j)\right] \\
&Both\ with \quad W(k) = \frac{w^{(k)}(t_0)}{k!}, \quad F(j) = \frac{f^{(j)}(t_0)}{j!} \quad \text{and} \quad G(k-j) = \frac{g^{(k-j)}(t_0)}{(k-j)!}
\end{aligned} \tag{5.2}$$

Here, both $f(t)$ and $g(t)$ are place-holder functions and $w(t)$ is called the auxiliary function which replaces the respective operation. Sometimes, the state derivative functions can be rather complex and have many operations intertwined. One can then choose to introduce extra variables to simplify the derivatives and reduce the number of operations in that particular derivative. These extra variables together with the expression that they replace then form the auxiliary equations. A random example is shown in Equation (5.3).

$$\begin{aligned}
x_4' &= u_4 = 2x_1 + \frac{(x_3x_1 + x_2x_3)}{x_4} \\
x_8 &= x_3x_1 + x_2x_3
\end{aligned} \tag{5.3}$$

However, now that the extra variable x_8 is introduced, the derivative of the corresponding auxiliary equations has to be described as well (see Equation (5.4), which in turn will use auxiliary functions to determine the k^{th} derivative of x_8.

$$\begin{aligned}
x_4' &= u_4 = 2x_1 + \frac{x_8}{x_4} \\
x_8' &= u_8 = x_3'x_1 + x_3x_1' + x_2'x_3 + x_2x_3' = u_3x_1 + x_3u_1 + u_2x_3 + x_2u_3
\end{aligned} \tag{5.4}$$

At this point it should be emphasised that using auxiliary equations and derivatives is a choice. It is not required to make TSI work. Sometimes it can be easier to introduce auxiliary equations because it makes the problem more comprehensible but this could introduce errors because all auxiliary equations also require auxiliary derivatives, which might not be easy to find. In those cases it could be more useful to just leave the state derivatives as they are and write more auxiliary functions instead. This all depends on the problem and the preference of the user.

Now let $u_n^{(k-1)} = x_n^k$ for $k = 1,\ldots,K$, then $\dfrac{u_n^{(k-1)}}{(k-1)!} = \dfrac{x_n^k}{(k-1)!}$, and also $\dfrac{x_n^k}{k!} = X_n(k)$. Combining this results in Equation (5.5).

$$U_n(k-1) = kX_n(k) \Rightarrow X_n(k) = \frac{U_n(k-1)}{k} \tag{5.5}$$

The $X_n(k)$ are also known as the Taylor Series coefficients of the n^{th} variable for the k^{th} order derivative. Also, $U_n(k)$ are the recurrence relation expressions for the n^{th} variable which consist of all the reduced auxiliary functions ($W(k)$) that form that derivative.

Using the expression described in Equation (5.5) all Taylor Series coefficients can be found. Then Equation (5.1) can be used to determine the next state values at time $t+h$ for the known previous parameter values at time t. The same can then be done to determine the values at the time-step after that using the state values at $t+h$, etc. Going through all these different time-steps results in the final integration.

5.1.2. STEP-SIZE

The step-size used for TSI can be either set as a constant value by the user or determined using a step-size controller, giving it the variable step-size properties, which is done in this thesis. The chosen step-size method was based on the recommendation of both Scott and Martini (2008) and Bergsma (2015) and directly uses the chosen (or preferred) local error tolerance τ to determine the next step size. This method assures that the step-size is small enough such that all variables satisfy the error condition directly. The step-size is determined using a so-called fixed-point iteration performed using Equation (5.6). The initial step-size h_1 can be

chosen to be the current step-size h as an easy estimate. Then the iteration is performed over l until a certain required convergence is reached.

$$h_{l+1} = exp\left(\frac{1}{K-1} ln\left[\frac{\tau}{\left|X_n(K-1)\right| + K\left|X_n(K)\right| h_l}\right]\right) \tag{5.6}$$

This is then done for all variables and the smallest required step-size is chosen, which is then used to determine the next step-size through $h_{next} = \eta h_{chosen}$, where η is the chosen step multiplication factor which has to be smaller than 1.

5.1.3. HANDLING MODEL SECTIONS

For both the temperature and drag coefficient models, different sections have been identified with different behaviours. Each of these sections was fitted with a polynomial approximation as described in Section 3.4. Because TSI only uses data available at the current point, it could be that if the step-size is allowed to be too large, the section boundary for one of these models is crossed. At that point, the step-size has to be adjusted such that it stops at the start of the next section. This way, the next time step will start at the next model section. This also means that these sections do not necessarily have to be connected, which means that TSI would be able to handle discontinuities in the model as well, should those occur.
This section boundary is found using a root finding method. The method used in this research is based on the method described in Bergsma (2015).

First, the simulation program has to determine if the newly computed parameter (either Mach number or altitude) has indeed moved into the new section. This can be done using the section limits as shown by Equation (5.7) which is called the limit function. From now on, the equations will be written for the altitude, but the same expressions hold for the Mach number.

$$f = h - h_{limit} \tag{5.7}$$

Because this research is based on an ascent trajectory, a new section is reached if the altitude is higher than the lower limit altitude of that section. Therefore, if $f > 0$ a new section has been reached, and the root finding method should be used to get the value of f back to zero and thus identify the step-size at which the root of f is located.

At the start of the root finding procedure, two points can be identified. The first point is the starting point of the MAV at the beginning of the integration step and the second point is the proposed end point of the MAV as suggested by the current step-size. If the new section has been reached it means that the point that the MAV is coming from has a limit function value that is below zero, or $f_{From} < 0$. The mission time at that point is given by t_{From}. This is the original mission time ($t_{original}$) at the start of the integration and it is therefore the goal to find h_{req} (required step-size) such that $t_{original} + h_{req}$ results in the final limit function value, $f_{new} = 0$. The mission time at the second point (where the MAV was headed when the section was crossed) is given by t_{To} with the corresponding $f_{To} > 0$. This means that f_{new} and t_{new} are located somewhere between these original points. The method described by Bergsma (2015) should result in a spiralling motion that slowly converges to the root. This means that the root finding method is always evaluated from the "from" point to the "to" point, irrespective of on which side of the root the next starting point is.

The first step is to determine the altitude derivative at the current "from" point, or $\dot{f}_{From}$. If the Taylor Series coefficients for the altitude (in this case) are available, then the derivative can directly be computed by combining Equations (5.1) and (5.5) resulting in Equation (5.8). Otherwise, the derivative has to be comprised of other known derivatives.

$$\dot{f}_{From} \approx \sum_{k=1}^{K} kX(k) h^{k-1} \tag{5.8}$$

Next, the coefficient of the root finding curve, α, has to be determined. This coefficient is effectively $\frac{\ddot{f}_{From}}{2}$ and is computed using Equation (5.9).

$$\alpha = \frac{f_{To} - f_{From}}{(t_{To} - t_{From})^2} - \frac{\dot{f}_{From}}{t_{To} - t_{From}} \tag{5.9}$$

With this coefficient it is possible to compute the new time of the newly computed point. This point is called the new from point because no matter if this point is before of after the root of the limit function, that point will be used as the starting or "from" point for the next point computation. This is why this time is called $t_{From,new}$ and is computed using Equation (5.10). This equation has a plus sign in front of the square root because the next section has been reached when $f > 0$.

$$t_{From,new} = t_{From} + \frac{\left(-\dot{f}_{From} + \sqrt{\dot{f}_{From}^2 + 4\alpha f_{From}}\right)}{2\alpha} \tag{5.10}$$

Now that the new time is known, the step-size can be computed by simply taking the difference between the original time and the new time. With this new step-size, the Taylor Series coefficients can be used as per Equation (5.1) to determine the new altitude at that new "from" point. This means that, using Equation (5.7) the new limit function value $f_{From,new} = h_{From,new}$-$h_{limit}$. If this $f_{From,new}$ value is zero it means that the section boundary has been found and the newly computed step-size is the step-size that should be used. However, if $f_{From,new}$ is on the other side of the root compared to f_{From} then the "from" and "to" points have to be updated. The conditions are shown in Equation (5.11).

$$\begin{aligned}
&\text{If the signs of } f_{From} \text{ and } f_{From,new} \text{ are oposite} \Rightarrow \begin{cases} f_{To} = f_{From} \\ t_{To} = t_{From} \\ f_{From} = f_{From,new} \\ t_{From} = t_{From,new} \end{cases} \\
&\text{If the signs of } f_{From} \text{ and } f_{From,new} \text{ are the same} \Rightarrow \begin{cases} f_{To} = f_{To} & \text{Remains the same} \\ t_{To} = t_{To} & \text{Remains the same} \\ f_{From} = f_{From,new} \\ t_{From} = t_{From,new} \end{cases}
\end{aligned} \tag{5.11}$$

With this, the root finding process can be started again at the first step (so starting at Equation (5.8)).

However, the condition that $f_{From,new}$ should be equal to zero before the root finding method is satisfied and the new step-size can be accepted is not realistic. This is why two conditions have to be met instead. $f_{From,new} \geq 0$ and $f_{From,new} \leq 1 \cdot 10^{-6}$. This means that $f_{From,new}$ has to be within an accuracy of 1 cm for the altitude and within a Mach accuracy of $1 \cdot 10^{-6}$. If both these cases are met, it is assured that the next section will be used in the next time step calculation, because the section limits have always been set at the beginning of the section.

5.2. ASSOCIATED EQUATIONS

In order for TSI to be implemented, the state derivatives have to be modelled as a set of continuous functions which are a function of the state only. This can be done in different ways. In this case, three cases were tested: two Cartesian cases and one spherical case. For the Cartesian cases, the initial conditions first have to be transformed into Cartesian coordinates. The Cartesian equations themselves require reference frame transformations, which can be written in two different ways. The first case in described in in Section 5.2.1 and the second in Section 5.2.2. The reason for testing the spherical case is that the initial conditions are provided in spherical coordinates and the intermediate computations can be easily interpreted and checked for errors. However, the equations are highly sensitive to singularities. The spherical equations are described in Section 5.2.3 and already include reference frame transformations.

5.2.1. CARTESIAN EQUATIONS, FIRST CASE

The Cartesian state is described in Equation (5.12), where m_{MAV} is the mass of the MAV and the subscript I refers to the inertial frame.

$$\mathbf{r} = \begin{pmatrix} x_I \\ y_I \\ z_I \end{pmatrix} \qquad \mathbf{V} = \begin{pmatrix} V_{x_I} \\ V_{y_I} \\ V_{z_I} \end{pmatrix} \qquad m_{MAV} \tag{5.12}$$

The corresponding state derivatives are then described by Equation (5.13).

$$\begin{aligned} \dot{x}_I &= V_{x_I} & \ddot{x}_I &= \dot{V}_{x_I} = a_{x_I} \\ \dot{y}_I &= V_{y_I} & \ddot{y}_I &= \dot{V}_{y_I} = a_{y_I} \\ \dot{z}_I &= V_{z_I} & \ddot{z}_I &= \dot{V}_{z_I} = a_{z_I} \end{aligned} \qquad \dot{m}_{MAV} = -\frac{T}{g_0 I_{sp}} \tag{5.13}$$

From this point on, the subscript I is omitted, because the state and the state derivatives are always presented in the inertial frame. If variables have to be presented in any other reference frame the corresponding subscripts will be provided and explained. The accelerations in the x-, y- and z-direction have three contributing components: gravitational acceleration, drag and thrust. The gravitational acceleration can be directly expressed in the inertial frame, however the drag is presented in the body frame and the thrust is expressed in the propulsion frame. Therefore, both the drag and thrust contributions have to be transformed to the inertial frame using transformation matrices. This then results in the expression for the acceleration vector as shown by Equation (5.14). The subscript G shows that the parameter is a function of the ground velocity.

$$\begin{aligned} \begin{pmatrix} a_x \\ a_y \\ a_z \end{pmatrix} = \begin{pmatrix} -\mu_M \frac{x}{r^3} \\ -\mu_M \frac{y}{r^3} \\ -\mu_M \frac{z}{r^3} \end{pmatrix} + \Big|_{\mathbf{I}} \mathbb{T}_{\mathbf{z}}(-\Omega_M t_O + \omega_P) \Big|_{\mathbf{R}} \mathbb{T}_{\mathbf{z}}(-\tau)\, \mathbb{T}_{\mathbf{y}}\left(\frac{\pi}{2} + \delta\right) \Big|_{\mathbf{V}} \mathbb{T}_{\mathbf{z}}(-\chi_G)\, \mathbb{T}_{\mathbf{y}}(-\gamma_G) \Big|_{\mathbf{B}} \left[\begin{pmatrix} -\frac{D}{m_{MAV}} \\ 0 \\ 0 \end{pmatrix} + \dots \right. \\ \left. \dots \Big|_{\mathbf{B}} \mathbb{T}_{\mathbf{z}}(-\psi_T)\, \mathbb{T}_{\mathbf{y}}(-\epsilon_T) \Big|_{\mathbf{P}} \begin{pmatrix} \frac{T}{m_{MAV}} \\ 0 \\ 0 \end{pmatrix} \right] \end{aligned} \tag{5.14}$$

It can be seen that this set of equations is a function of the current position and many other parameters. These parameters will all have to be written as a function of the current state. This can be done by writing them into auxiliary equations, forming extra variables that then also require the auxiliary derivatives. This works for certain parameters, such as the gravity, because these equations are already expressed in the inertial frame. However, the transformation angles are defined in different reference frames, which means that finding the proper auxiliary derivatives can be tricky sometimes. Therefore it was decided to directly write these parameters as auxiliary functions. Each of the auxiliary functions performs one simple algebraic operation and the collection of these auxiliary functions then form the complete set of recurrence relations using the recurrence relations for the simple algebraic operations. For the auxiliary equations, a similar notation will be used as shown by Scott and Martini (2008). Here the equations are denoted by x_{number} and the corresponding derivatives x'_{number}. This notation will also be used for the current state and the corresponding state derivatives. This way, Equation (5.12) can be written as Equation (5.15) and the corresponding derivatives can be written as presented by Equation (5.16).

$$\begin{aligned} x_1 &= x & x_4 &= \dot{x} = V_x \\ x_2 &= y & x_5 &= \dot{y} = V_y \\ x_3 &= z & x_6 &= \dot{z} = V_z \end{aligned} \qquad x_7 = m_{MAV} \tag{5.15}$$

$$\begin{aligned} x'_1 &= x_4 & x'_4 &= \dot{V}_x = a_x = a_{g,x} + a_{D,x} + a_{T,x} \\ x'_2 &= x_5 & x'_5 &= \dot{V}_y = a_y = a_{g,y} + a_{D,y} + a_{T,y} \\ x'_3 &= x_6 & x'_6 &= \dot{V}_z = a_z = a_{g,z} + a_{D,z} + a_{T,z} \end{aligned} \qquad x'_7 = \dot{m}_{MAV} = -\frac{T}{g_0 I_{sp}} \tag{5.16}$$

In this case the thrust and specific impulse are constant, which means that x'_7 is constant and any additional derivative will be zero. Also, neither one of the thrust angles is a function of the state, which means

that a_T, in the body frame, is only a function of x_7 (also see Equation (5.14)). However, both a_g and a_D are a function of the position and velocity, where a_D is also a function of the MAV mass. Only a_g is rewritten using auxiliary equations as mentioned before.

GRAVITATIONAL ACCELERATION

For the gravitational acceleration two auxiliary equations were required since $r = \sqrt{x^2 + y^2 + z^2}$. The resulting expressions for the gravitational acceleration are shown in Equation (5.17) with the corresponding auxiliary equations and the derivatives defined in Equation (5.18).

$$\begin{aligned} a_{g,x} &= -\mu_M \frac{x_1}{r^3} = \frac{x_1}{\left(x_1^2 + x_2^2 + x_3^2\right)^{3/2}} = -\mu_M \frac{x_1}{x_9} \\ a_{g,y} &= -\mu_M \frac{x_2}{r^3} = \frac{x_2}{\left(x_1^2 + x_2^2 + x_3^2\right)^{3/2}} = -\mu_M \frac{x_2}{x_9} \\ a_{g,z} &= -\mu_M \frac{x_3}{r^3} = \frac{x_3}{\left(x_1^2 + x_2^2 + x_3^2\right)^{3/2}} = -\mu_M \frac{x_3}{x_9} \end{aligned} \tag{5.17}$$

$$\begin{aligned} x_8 &= x_1^2 + x_2^2 + x_3^2 & x_8' &= 2x_1x_4 + 2x_2x_5 + 2x_3x_6 \\ x_9 &= x_8^{3/2} & x_9' &= \frac{3}{2}\frac{x_9 x_8'}{x_8} \end{aligned} \tag{5.18}$$

TRANSFORMATION ANGLES

The angles required for the transformation to go from the body frame to the reference frame all have to be written as a function of the state variables. The required angles are λ, δ, χ_G and γ_G. Here $\lambda = \tau + \Omega_M t_O - \omega_P$. This means that instead of first transforming to the rotating frame completely and then transforming to the inertial frame, an inertial longitude angle λ can be defined to directly transform to the inertial frame. The first two angles are the longitude and latitude and are spherical coordinates. Thus the relations for these angles can be found using the transformation from the Cartesian to the spherical system. However, the angles themselves are not required directly, because they are only used in transformation matrices. These matrices are comprised of the sines and cosines of these angles, which means that a direct relation between the state variables and the sines and cosines of the position angles can be used. These relations can be derived from Figure 5.1 and are described in Equation (5.19)

$$\begin{aligned} r &= \sqrt{x^2 + y^2 + z^2} & \sin\lambda &= \frac{y}{\sqrt{x^2 + y^2}} & \sin\delta &= \frac{z}{r} \\ & & \cos\lambda &= \frac{x}{\sqrt{x^2 + y^2}} & \cos\delta &= \frac{\sqrt{x^2 + y^2}}{r} \end{aligned} \tag{5.19}$$

The corresponding auxiliary functions can then be described by Equation (5.20) using the definitions provided in Equations (5.15) and (5.16).

$$\begin{aligned} w_{4,1} &= x_1^2 + x_2^2 & w_{4,5} &= s\lambda = \frac{x_2}{w_{4,4}} & w_{4,7} &= s\delta = \frac{x_3}{w_{4,3}} \\ w_{4,2} &= w_{4,1} + x_3^2 & w_{4,6} &= c\lambda = \frac{x_1}{w_{4,4}} & w_{4,8} &= c\delta = \frac{w_{4,4}}{w_{4,3}} \\ w_{4,3} &= r = \sqrt{w_{4,2}} & & & & \\ w_{4,4} &= \sqrt{w_{4,1}} & & & & \end{aligned} \tag{5.20}$$

The latitude and longitude could be described using the position vector in the inertial frame, however the transformation from the body frame to the vertical frame is a function of the ground (underscore 'G') velocity in the rotational frame. Since the position and velocity in the inertial frame are known (current state), the ground velocity (V_G) can be written as a function of the inertial velocity (V_I). For this, the velocity components have to be transformed from the inertial frame to the vertical frame (which is the inverse of the first three transformations described in Equation (5.14)). This transformation is shown in Equation (5.21) and was described in Mooij (1994). Here, c stands for cosine and s stands for sine. This transformation also includes the rotational effect on the velocity due to the rotation of Mars.

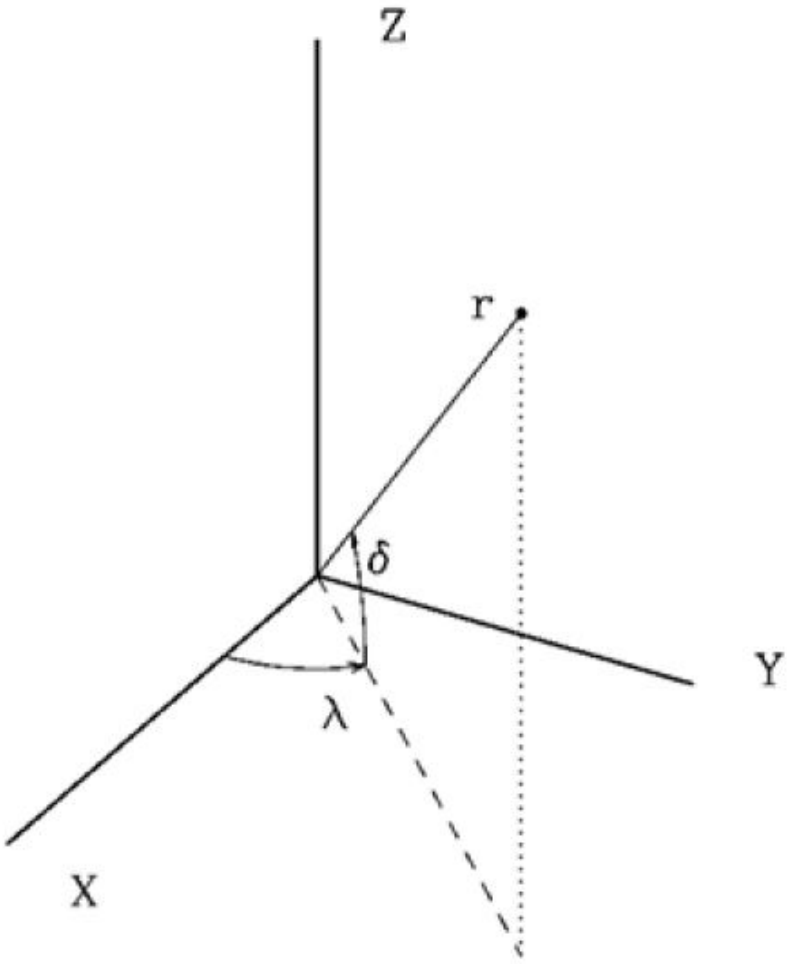


Figure 5.1: Spherical position variables in an inertial Cartesian frame (Noomen, 2013a).

$$\mathbf{V_V} = \begin{pmatrix} V_{x_V} \\ V_{y_V} \\ V_{z_V} \end{pmatrix} = \begin{bmatrix} -c\lambda s\delta & -s\lambda s\delta & c\delta \\ -s\lambda & c\lambda & 0 \\ -c\lambda c\delta & -s\lambda c\delta & -s\delta \end{bmatrix} \left\{ \begin{pmatrix} V_x \\ V_y \\ V_z \end{pmatrix} - \begin{pmatrix} 0 \\ 0 \\ \Omega_M \end{pmatrix} \times \begin{pmatrix} x \\ y \\ z \end{pmatrix} \right\} \tag{5.21}$$

The ground velocity can now be computed as the norm of the vertical velocity vector as shown by Equation (5.22). The transformation matrices disappear because the norm of a transformation matrix is simply 1, which means that $V_G = V_V = V_R$.

$$V_G = \|\mathbf{V_V}\| = \left\| \begin{pmatrix} V_x \\ V_y \\ V_z \end{pmatrix} - \begin{pmatrix} 0 \\ 0 \\ \Omega_M \end{pmatrix} \times \begin{pmatrix} x \\ y \\ z \end{pmatrix} \right\| \tag{5.22}$$

Equation (5.22) can be rewritten as Equation (5.23).

$$V_G = \sqrt{\left(V_x + \Omega_M y\right)^2 + \left(V_y - \Omega_M x\right)^2 + V_z^2} \tag{5.23}$$

The corresponding auxiliary functions are provided in Equation (5.24).

$$\begin{aligned} w_{4,9} &= V_x + \Omega_M y = x_4 + \Omega_M x_2 & \qquad w_{4,11} &= w_{4,9}^2 + w_{4,10}^2 + x_6^2 \\ w_{4,10} &= V_y - \Omega_M x = x_5 - \Omega_M x_1 & \qquad w_{4,12} &= V_G = \sqrt{w_{4,11}} \end{aligned} \tag{5.24}$$

The spherical velocity angles can now be derifed from Figure 5.2 as described by Equation (5.25). These were also provided by Mooij (1994).

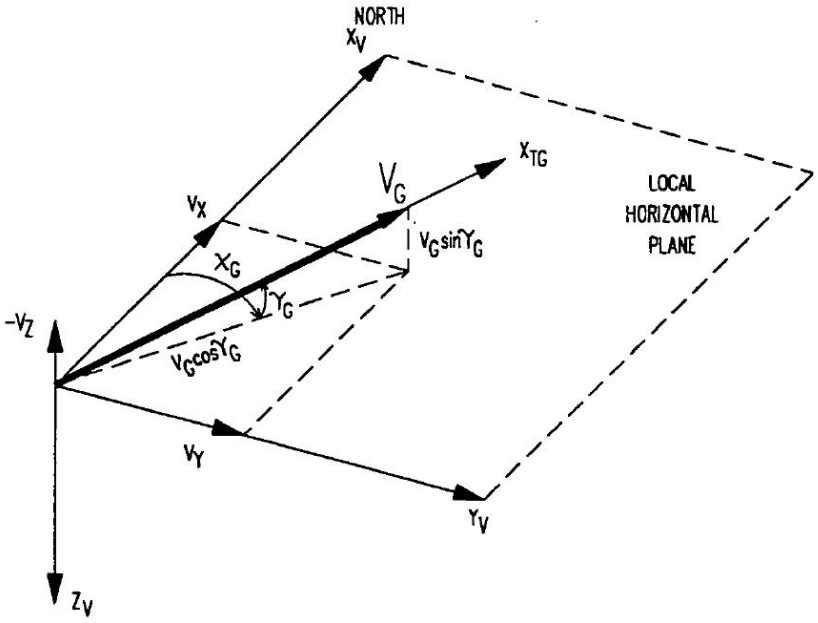


Figure 5.2: Spherical velocity variables in a vertical Cartesian frame (Mooij, 1994).

$$\begin{aligned} \sin\chi_G &= \frac{V_{y_V}}{\sqrt{V_{x_V}^2 + V_{y_V}^2}} & \sin\gamma_G &= \frac{-V_{z_V}}{V_G} \\ \cos\chi_G &= \frac{V_{x_V}}{\sqrt{V_{x_V}^2 + V_{y_V}^2}} & \cos\gamma_G &= \frac{\sqrt{V_{x_V}^2 + V_{y_V}^2}}{V_G} \end{aligned} \tag{5.25}$$

Here, V_{x_V}, V_{y_V} and V_{z_V} are the velocities in the vertical frame. Expressions for these variables can be obtained by rewriting Equation (5.21). This then results in Equation (5.27).

$$\begin{aligned} V_{x_V} &= -\left(V_x + \Omega_M y\right) s\delta c\lambda - \left(V_y - \Omega_M x\right) s\delta s\lambda + V_z c\delta \\ V_{y_V} &= \left(V_y - \Omega_M x\right) c\lambda - \left(V_x + \Omega_M y\right) s\lambda \\ V_{z_V} &= -\left(V_x + \Omega_M y\right) c\delta c\lambda - \left(V_y - \Omega_M x\right) c\delta s\lambda - V_z s\delta \end{aligned} \tag{5.26}$$

The combined auxiliary functions for Equations (5.25) and (5.27) are described in Equation (5.24).

$$\begin{aligned} w_{4,13} &= -c\lambda s\delta = -w_{4,6} w_{4,7} \\ w_{4,14} &= -s\delta s\lambda = -w_{4,7} w_{4,5} \\ w_{4,15} &= -c\delta c\lambda = -w_{4,8} w_{4,6} \\ w_{4,16} &= -c\delta s\lambda = w_{4,8} w_{4,5} \\ w_{4,17} &= V_{x_V} = x_6 w_{4,8} + w_{4,9} w_{4,13} + w_{4,10} w_{4,14} \\ w_{4,18} &= V_{y_V} = w_{4,10} w_{4,6} - w_{4,9} w_{4,5} \\ w_{4,19} &= V_{z_V} = w_{4,9} w_{4,15} - x_6 w_{4,7} + w_{4,10} w_{4,16} \\ w_{4,20} &= V_{x_V}^2 + V_{y_V}^2 = w_{4,17}^2 + w_{4,18}^2 \\ w_{4,21} &= \sqrt{w_{4,20}} \\ w_{4,22} &= s\chi_G = \frac{w_{4,18}}{w_{4,21}} \\ w_{4,23} &= c\chi_G = \frac{w_{4,17}}{w_{4,21}} \\ w_{4,24} &= s\gamma_G = -\frac{w_{4,19}}{w_{4,12}} \\ w_{4,25} &= c\gamma_G = \frac{w_{4,21}}{w_{4,12}} \end{aligned} \tag{5.27}$$

DRAG ACCELERATION

The drag acceleration is determined in the body frame by dividing the drag force (D) by the mass of the MAV (x_7). The drag force itself is a function of the position and velocity. The equations associated with the drag function are described in Equation (5.28) except for the C_D equations. The polynomial coefficients for the density equation are provided in Table 3.4 and are represented in Equation (5.28) by P_ρ.

$$\begin{aligned} h &= r - R_{MOLA} \\ D &= \frac{1}{2}\rho V_G^2 S C_D \\ \rho &= e^{P_{\rho 10}h^{10}+P_{\rho 9}h^9+P_{\rho 8}h^8+P_{\rho 7}h^7+P_{\rho 6}h^6+P_{\rho 5}h^5+P_{\rho 4}h^4+P_{\rho 3}h^3+P_{\rho 2}h^2+P_{\rho 1}h+P_{\rho 0}} \end{aligned} \tag{5.28}$$

The numbering for the drag auxiliary functions start with 27 because it was added later on and because it used to be an auxiliary equation. The auxiliary functions for the density can then be described by Equation (5.34).

$$\begin{aligned} w_{27,1} &= h = w_{4,3} - R_{MOLA} & w_{27,7} &= w_{27,1}^7 \\ w_{27,2} &= w_{27,1}^2 & w_{27,8} &= w_{27,1}^8 \\ w_{27,3} &= w_{27,1}^3 & w_{27,9} &= w_{27,1}^9 \\ w_{27,4} &= w_{27,1}^4 & w_{27,10} &= w_{27,1}^{10} \\ w_{27,5} &= w_{27,1}^5 & w_{27,11} &= P_{\rho 10}w_{27,10} + P_{\rho 9}w_{27,9} + \ldots + P_{\rho 1}w_{27,1} + P_{\rho 0} \\ w_{27,6} &= w_{27,1}^6 & w_{27,12} &= \rho = e^{w_{27,11}} \end{aligned} \tag{5.29}$$

Since the drag coefficient is a function of Mach number as by Figure 2.4 and is not a continuous function, it has to be split into 6 different sections. Each section has a separate $C_D - M$ relation. Before these relations can be described, three additional expressions are required which are described in Equation (5.30).

$$\begin{aligned} M &= \frac{V_G}{a} \\ a &= \sqrt{\gamma_a R_a^* T_a} \quad \text{where} \quad R_a^* = \frac{R_a}{M_a} \end{aligned} \tag{5.30}$$

Where the corresponding auxiliary functions can be described by Equation (5.31).

$$\begin{aligned} w_{27,14} &= a = \sqrt{\gamma_a R_a^* w_{27,13}} \\ w_{27,15} &= M = \frac{w_{4,12}}{w_{27,14}} \end{aligned} \tag{5.31}$$

The conditional relations shown in Equation (5.32) describe the different equations that have to be used associated with the different sections of the drag coefficient plot. Here $P_{C_D number,section}$ are the polynomial fit coefficients as provided in Table 3.5.

$$C_D = \begin{cases} 0.2, & \text{for } 0 \leq M < 0.5 \\ P_{C_D 1,2}M + P_{C_D 0,2}, & \text{for } 0.5 \leq M < 1 \\ P_{C_D 1,3}M + P_{C_D 0,3}, & \text{for } 1 \leq M < 1.3 \\ P_{C_D 1,4}M + P_{C_D 0,4}, & \text{for } 1.3 \leq M < 2.5 \\ P_{C_D 1,5}M + P_{C_D 0,5}, & \text{for } 2.5 \leq M < 4 \\ 0.3, & \text{for } M \geq 4 \end{cases} \tag{5.32}$$

In this case, the auxiliary functions for $C_D\,(= w_{27,16})$ is any of the conditional relations depending on $M\,(= w_{27,15})$.

This only leaves the temperature $T_a\,(= w_{27,13})$, which is a function of the altitude $h\,(= w_{27,1})$ in km,MOLA. But as described in Section 3.4.1, this parameter is split into different sections as well. The equations per section for the temperature is provided in Equation (5.33). Here $P_{T number,section}$ are the polynomial fit coefficients as provided in Table 3.2.

$$T_a = \begin{cases} P_{T1,1}h + P_{T0,1}, & \text{for } -0.6 \leq h < 5.04 \\ P_{T3,2}h^3 + P_{T2,2}h^2 + P_{T1,2}h + P_{T0,2}, & \text{for } 5.04 \leq h < 35.53 \\ P_{T6,3}h^6 + P_{T5,3}h^5 + P_{T4,3}h^4 + P_{T3,3}h^3 + \ldots & \\ \quad \ldots + P_{T2,3}h^2 + P_{T1,3}h + P_{T0,3}, & \text{for } 35.53 \leq h < 75.07 \\ P_{T8,4}h^8 + P_{T7,4}h^7 + P_{T6,4}h^6 + P_{T5,4}h^5 + \ldots & \\ \quad \ldots + P_{T4,4}h^4 + P_{T3,4}h^3 + P_{T2,4}h^2 + P_{T1,4}h + P_{T0,4}, & \text{for } 75.07 \leq h < 170.05 \\ 136.5, & \text{for } h \geq 170.05 \end{cases} \tag{5.33}$$

For both the conditional parameters C_D and T_a the required section has to be determined before the evaluation of the equations.

The drag can now also be described as an auxiliary function as shown by Equation (5.34).

$$\begin{aligned} w_{27,17} &= V_G^2 = w_{4,12}^2 \\ w_{27,18} &= V_G^2 C_D = w_{27,17}\, w_{27,16} \\ w_{27,19} &= D = \frac{1}{2} S w_{27,18}\, w_{27,12} \end{aligned} \tag{5.34}$$

THRUST ACCELERATION

The only acceleration component still missing is the thrust acceleration. To be able to write the auxiliary functions for the thrust, Equation (5.14) first has to be rewritten such that all the transformations are gathered into two transformation matrices (see Equation (5.35)). The transformation matrices are described by Equations (5.36) and (5.37) for $\mathbb{T}_{\mathbf{BP}}$ and $\mathbb{T}_{\mathbf{IB}}$ respectively.

$$\begin{pmatrix} x_4' \\ x_5' \\ x_6' \end{pmatrix} = \begin{pmatrix} -\mu_M \dfrac{x_1}{x_9} \\ -\mu_M \dfrac{x_2}{x_9} \\ -\mu_M \dfrac{x_3}{x_9} \end{pmatrix} + \mathbb{T}_{\mathbf{IB}} \left[\begin{pmatrix} -\dfrac{w_{27,19}}{x_7} \\ 0 \\ 0 \end{pmatrix} + \mathbb{T}_{\mathbf{BP}} \begin{pmatrix} \dfrac{T}{x_7} \\ 0 \\ 0 \end{pmatrix} \right] \tag{5.35}$$

$$\mathbb{T}_{\mathbf{BP}} = \begin{bmatrix} c\psi_T c\epsilon_T & -s\psi_T & c\psi_T s\epsilon_T \\ s\psi_T c\epsilon_T & c\psi_T & s\psi_T s\epsilon_T \\ -s\epsilon_T & 0 & c\epsilon_T \end{bmatrix} \tag{5.36}$$

$$\mathbb{T}_{\mathbf{IB}} = \begin{bmatrix} c\lambda\left(-s\delta c\chi c\gamma + c\delta s\gamma\right) - s\lambda s\chi c\gamma & c\lambda s\delta s\chi - s\lambda c\chi & c\lambda\left(-s\delta c\chi s\gamma - c\delta c\gamma\right) - s\lambda s\chi s\gamma \\ s\lambda\left(-s\delta c\chi c\gamma + c\delta s\gamma\right) + c\lambda s\chi c\gamma & s\lambda s\delta s\chi + c\lambda c\chi & s\lambda\left(-s\delta c\chi s\gamma - c\delta c\gamma\right) + c\lambda s\chi s\gamma \\ c\delta c\chi c\gamma + s\delta s\gamma & -c\delta s\chi & c\delta c\chi s\gamma - s\delta c\gamma \end{bmatrix} \tag{5.37}$$

The thrust accelerations in the x-, y- and z-directions (in the body frame) can now be found by rewriting the last part of Equation (5.35) to Equation (5.38) using Equation (5.36).

$$\mathbb{T}_{\mathbf{BP}} \begin{pmatrix} \dfrac{T}{x_7} \\ 0 \\ 0 \end{pmatrix} = \begin{pmatrix} \dfrac{T}{x_7} c\psi_T c\epsilon_T \\ \dfrac{T}{x_7} s\psi_T c\epsilon_T \\ -\dfrac{T}{x_7} s\epsilon_T \end{pmatrix} \tag{5.38}$$

Equation (5.38) can be expressed as a collection of auxiliary functions as well. These are described in Equation (5.39).

$$
\begin{aligned}
w_{4,26} &= \cos\psi_T & w_{4,33} &= \frac{T}{x_7} = T w_{4,32} \\
w_{4,27} &= \cos\epsilon_T & w_{4,34} &= \frac{T}{x_7} c\epsilon_T c\psi_T = w_{4,33} w_{4,30} \\
w_{4,28} &= \sin\psi_T & w_{4,37} &= \frac{T}{x_7} c\epsilon_T s\psi_T = w_{4,33} w_{4,31} \\
w_{4,29} &= \sin\epsilon_T & w_{4,38} &= \frac{T}{x_7} s\epsilon_T = w_{4,33} w_{4,29} \\
w_{4,30} &= c\psi_T c\epsilon_T = w_{4,26} w_{4,27} & & \\
w_{4,31} &= c\epsilon_T s\psi_T = w_{4,27} w_{4,28} & & \\
w_{4,32} &= \frac{1}{x_7} & &
\end{aligned}
\tag{5.39}
$$

The thrust accelerations are now written in the body frame, which means that they can be combined with the drag acceleration in the body frame. This is done in Equation (5.40). Here, $w_{4,35}$ represents the drag acceleration in the body frame and $w_{4,36}$ is the total acceleration in the x-direction in the body frame caused by both the drag and the thrust.

$$
\begin{pmatrix} -\frac{x_{27}}{x_7} \\ 0 \\ 0 \end{pmatrix} + \begin{pmatrix} w_{4,34} \\ w_{4,37} \\ -w_{4,38} \end{pmatrix} = \begin{pmatrix} w_{4,34} - w_{4,35} \\ w_{4,37} \\ -w_{4,38} \end{pmatrix} = \begin{pmatrix} w_{4,36} \\ w_{4,37} \\ -w_{4,38} \end{pmatrix}
\tag{5.40}
$$

ALL ACCELERATIONS COMBINED

At this point the gravity accelerations are known in the inertial frame and the drag and thrust accelerations in the body frame. In order to get the final acceleration vector in the inertial frame, the drag and thrust accelerations have to be transformed from the body frame to the inertial frame using $\mathbb{T}_{\mathbf{IB}}$ resulting in Equation (5.41).

$$
\mathbb{T}_{\mathbf{IB}} \begin{pmatrix} w_{4,36} \\ w_{4,37} \\ -w_{4,38} \end{pmatrix} = \begin{pmatrix} w_{4,36}\left(c\lambda\left(-s\delta c\chi c\gamma + c\delta s\gamma\right) - s\lambda s\chi c\gamma\right) + w_{4,37}\left(c\lambda s\delta s\chi - s\lambda c\chi\right) - w_{4,38}\left(c\lambda\left(-s\delta c\chi s\gamma - c\delta c\gamma\right) - s\lambda s\chi s\gamma\right) \\ w_{4,36}\left(s\lambda\left(-s\delta c\chi c\gamma + c\delta s\gamma\right) + c\lambda s\chi c\gamma\right) + w_{4,37}\left(s\lambda s\delta s\chi + c\lambda c\chi\right) - w_{4,38}\left(s\lambda\left(-s\delta c\chi s\gamma - c\delta c\gamma\right) + c\lambda s\chi s\gamma\right) \\ w_{4,36}\left(c\delta c\chi c\gamma + s\delta s\gamma\right) + w_{4,37}\left(-c\delta s\chi\right) - w_{4,38}\left(c\delta c\chi s\gamma - s\delta c\gamma\right) \end{pmatrix}
\tag{5.41}
$$

Now including the gravity components as well, the (lengthy) expressions for x_4', x_5' and x_6' become Equations (5.42) to (5.44) respectively.

$$
\begin{aligned}
x_4' = -\mu_M \frac{x_1}{x_9} &+ w_{4,36}\left(c\lambda\left(-s\delta c\chi c\gamma + c\delta s\gamma\right) - s\lambda s\chi c\gamma\right) + \ldots \\
&\ldots w_{4,37}\left(c\lambda s\delta s\chi - s\lambda c\chi\right) - \ldots \\
&\ldots w_{4,38}\left(c\lambda\left(-s\delta c\chi s\gamma - c\delta c\gamma\right) - s\lambda s\chi s\gamma\right)
\end{aligned}
\tag{5.42}
$$

$$
\begin{aligned}
x_5' = -\mu_M \frac{x_2}{x_9} &+ w_{4,36}\left(s\lambda\left(-s\delta c\chi c\gamma + c\delta s\gamma\right) + c\lambda s\chi c\gamma\right) + \ldots \\
&\ldots w_{4,37}\left(s\lambda s\delta s\chi + c\lambda c\chi\right) - \ldots \\
&\ldots w_{4,38}\left(s\lambda\left(-s\delta c\chi s\gamma - c\delta c\gamma\right) + c\lambda s\chi s\gamma\right)
\end{aligned}
\tag{5.43}
$$

$$
x_6' = -\mu_M \frac{x_3}{x_9} + w_{4,36}\left(c\delta c\chi c\gamma + s\delta s\gamma\right) + w_{4,37}\left(-c\delta s\chi\right) - w_{4,38}\left(c\delta c\chi s\gamma - s\delta c\gamma\right)
\tag{5.44}
$$

These equations now have to be written as a collection of auxiliary functions for the x-, y- and z-direction in the inertial frame. The gravitational acceleration can be written as the vector shown by Equation (5.45).

$$
\mathbf{a_g} = \begin{pmatrix} -\mu_M \frac{x_1}{x_9} \\ -\mu_M \frac{x_2}{x_9} \\ -\mu_M \frac{x_3}{x_9} \end{pmatrix} = \begin{pmatrix} w_{4,39} \\ w_{5,1} \\ w_{6,1} \end{pmatrix}
\tag{5.45}
$$

The auxiliary derivatives ($u = x'$) can now be defined as the collection of auxiliary functions that describe the different transformations for $u_4, u_5 and u_6$. These are shown in Equations (5.46) to (5.48) respectively.

$$\begin{aligned}
w_{4,40} &= -s\delta c\chi_G = -w_{4,7}\,w_{4,23} & w_{4,46} &= w_{4,42}\,c\gamma_G = w_{4,42}\,w_{4,25}\\
w_{4,41} &= c\delta s\gamma_G = w_{4,8}\,w_{4,24} & w_{4,47} &= -w_{4,13}\,s\chi_G = -w_{4,13}\,w_{4,22}\\
w_{4,42} &= -s\lambda s\chi_G = -w_{4,5}\,w_{4,22} & w_{4,48} &= w_{4,40}\,s\gamma_G = w_{4,40}\,w_{4,24}\\
w_{4,43} &= -s\lambda c\chi_G = -w_{4,5}\,w_{4,23} & w_{4,49} &= w_{4,42}\,s\gamma_G = w_{4,42}\,w_{4,24}\\
w_{4,44} &= -c\delta c\gamma_G = -w_{4,8}\,w_{4,25} & w_{4,50} &= c\lambda\left(w_{4,45}+w_{4,41}\right)+w_{4,46} = w_{4,6}\left(w_{4,45}+w_{4,41}\right)+w_{4,46}\\
w_{4,45} &= w_{4,40}\,c\gamma_G = w_{4,40}\,w_{4,25} & w_{4,51} &= c\lambda\left(w_{4,48}+w_{4,44}\right)+w_{4,49} = w_{4,6}\left(w_{4,48}+w_{4,44}\right)+w_{4,49}\\
& & w_{4,52} &= w_{4,39}+w_{4,36}\,w_{4,50}+w_{4,37}\left(w_{4,47}+w_{4,43}\right)-w_{4,38}\,w_{4,51}\\
& & u_4 &= w_{4,52}
\end{aligned} \tag{5.46}$$

$$\begin{aligned}
w_{5,2} &= c\lambda s\chi_G = w_{4,6}\,w_{4,22}\\
w_{5,3} &= s\lambda\left(w_{4,45}+w_{4,41}\right)+w_{5,2}\,c\gamma_G = w_{4,5}\left(w_{4,45}+w_{4,41}\right)+w_{5,2}\,w_{4,25}\\
w_{5,4} &= -w_{4,14}\,s\chi_G + c\lambda c\chi_G = -w_{4,14}\,w_{4,22}+w_{4,6}\,w_{4,23}\\
w_{5,5} &= s\lambda\left(w_{4,48}+w_{4,44}\right)+w_{5,2}\,s\gamma_G = w_{4,5}\left(w_{4,48}+w_{4,44}\right)+w_{5,2}\,w_{4,24}\\
w_{5,6} &= w_{5,1}+w_{4,36}\,w_{5,3}+w_{4,37}\,w_{5,4}-w_{4,38}\,w_{5,5}\\
u_5 &= w_{5,6}
\end{aligned} \tag{5.47}$$

$$\begin{aligned}
w_{6,2} &= s\delta s\gamma_G = w_{4,7}\,w_{4,24} & w_{6,5} &= -w_{4,44}\,c\chi_G + w_{6,2} = -w_{4,44}\,w_{4,23}+w_{6,2}\\
w_{6,3} &= c\delta s\chi_G = w_{4,8}\,w_{4,22} & w_{6,6} &= w_{4,41}\,c\chi_G + w_{6,4} = w_{4,41}\,w_{4,23}+w_{6,4}\\
w_{6,4} &= -s\delta c\gamma_G = -w_{4,7}\,w_{4,25} & w_{6,7} &= w_{6,1}+w_{4,36}\,w_{6,5}-w_{4,37}\,w_{6,3}-w_{4,38}\,w_{6,6}\\
& & u_6 &= w_{6,7}
\end{aligned} \tag{5.48}$$

Also, because auxiliary equations were used for the gravitational acceleration. The corresponding auxiliary derivatives can be written as a collection of auxiliary functions as shown in Equation (5.49).

$$\begin{aligned}
w_{8,1} &= x_1\,x_4 & w_{9,0} &= x_9\,u_8\\
w_{8,2} &= x_2\,x_5 & w_{9,1} &= \frac{w_{9,0}}{x_8}\\
w_{8,3} &= x_3\,x_6 & u_9 &= 1.5\,w_{9,1}\\
u_8 &= 2\left(w_{8,1}+w_{8,2}+w_{8,3}\right)
\end{aligned} \tag{5.49}$$

Then the equations in Equation (5.50) complete the whole set of auxiliary derivatives.

$$\begin{aligned}
u_1 &= x_4 & u_3 &= x_6\\
u_2 &= x_5 & u_7 &= -\frac{T}{g_0 I_{sp}}
\end{aligned} \tag{5.50}$$

5.2.2. CARTESIAN EQUATIONS, SECOND CASE

The second case (based on a method described by (Bergsma, 2015)) is similar to the first case in terms of set-up of the dynamic equations, the gravitational acceleration, and the drag and thrust accelerations in the body frame. The only difference lies in the formulation of $\mathbb{T}_{\mathbf{IB}}$. Instead of using Euler angles and rotation transformation matrices, $\mathbb{T}_{\mathbf{IB}}$ can be set-up as one single matrix performing a direct transformation. This matrix is described in Equation (5.51), where C stands for transformation matrix coefficient.

$$\mathbb{T}_{\mathbf{IB}} = \begin{bmatrix} C_1 & C_4 & C_7 \\ C_2 & C_5 & C_8 \\ C_3 & C_6 & C_9 \end{bmatrix} \tag{5.51}$$

As was seen in Section 5.2.1 the rotation from the vertical to the inertial frame is purely a function of the radius in the inertial frame ($\mathbf{r}$ or $\mathbf{r_I}$) Also, the transformation from the body frame to the vertical frame was purely a function of the ground velocity in the vertical frame ($\mathbf{V_V}$ or $\mathbf{V_{G_V}}$). Therefore, if this transformation is to be done in one transformation, the ground velocity in the inertial frame is required ($\mathbf{V_{G_I}}$), because it would transform directly to the inertial frame. This means that the coefficients in Equation (5.51) will have to be a function of $\mathbf{r_I}$ and $\mathbf{V_{G_I}}$, where $\mathbf{V_{G_I}}$ is given by Equation (5.52).

$$\mathbf{V_{G_I}} = \begin{pmatrix} V_x \\ V_y \\ V_z \end{pmatrix} - \begin{pmatrix} 0 \\ 0 \\ \Omega_M \end{pmatrix} \times \begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} V_x + \Omega_M y \\ V_y - \Omega_M x \\ V_z \end{pmatrix} \tag{5.52}$$

BODY FRAME UNIT VECTORS

So when looking from the inertial frame, there are two state vectors: position and ground velocity. The state changes part due to the accelerations in the body frame $\mathbf{a_B}$, which is a vector comprised of the drag and thrust accelerations. This vector has the same length and direction in the body frame and the inertial frame. Also, any vector is simply a multiplication of the corresponding unit vector. The same holds for the velocity, which means that because the velocity in the body frame is defined through the x-axis of that frame, the unit vector of the ground velocity can be described as the unit vector in the x-direction $\hat{\mathbf{i}}$ as shown by Equation (5.53).

$$\hat{\mathbf{i}} = \frac{\mathbf{V_{G_I}}}{\|\mathbf{V_{G_I}}\|} \tag{5.53}$$

This unit vector is also shown in Figure 5.3.

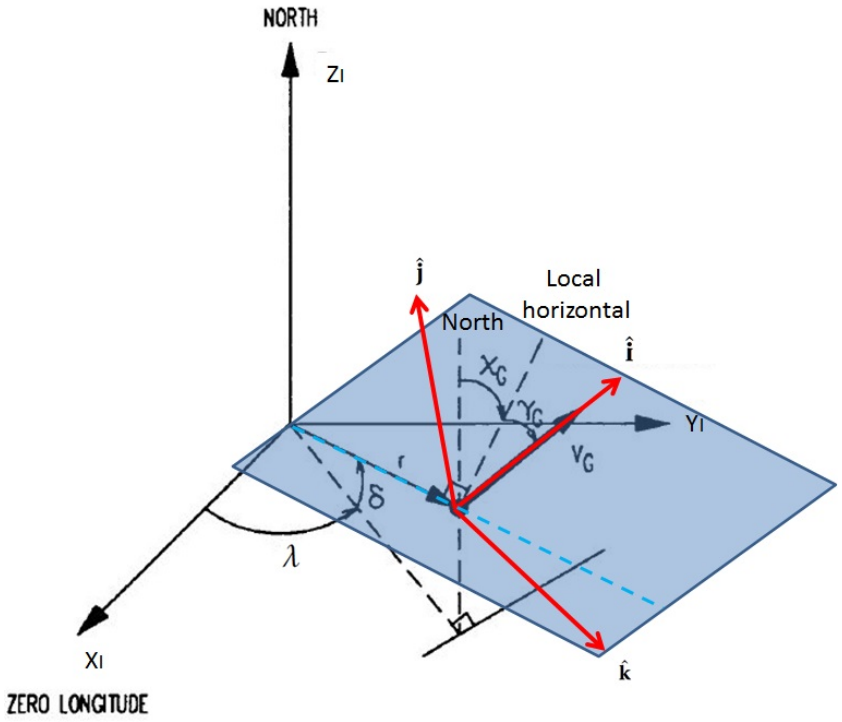


Figure 5.3: Definition of the body frame unit vectors based on the radius and ground velocity expressed in the inertial frame. (Mooij, 1994)

In Figure 5.3 it can be seen that the z-axis unit vector $\hat{\mathbf{k}}$ is defined in the same plane as $\hat{\mathbf{i}}$, $\mathbf{V_{G_I}}$ and $\mathbf{r_I}$ perpendicular to $\hat{\mathbf{i}}$. As a matter of fact, if the flight path angle is zero degrees, $\hat{\mathbf{k}}$ points in the same direction as the radius vector. In that case, the y-axis unit vector $\hat{\mathbf{j}}$ completes the unit frame. Since $\hat{\mathbf{j}}$ is perpendicular to the $\hat{\mathbf{i}}$-$\hat{\mathbf{k}}$ plane, it is also perpendicular to $\mathbf{V_{G_I}}$ and $\mathbf{r_I}$. Now because the radius vector is pointing through the centre of the body away from the centre of the inertial frame, the Right-hand-rule (RHR) gives Equation (5.54).

$$\hat{\mathbf{j}} = \frac{\mathbf{r_I} \times \mathbf{V_{G_I}}}{\|\mathbf{r_I} \times \mathbf{V_{G_I}}\|} \tag{5.54}$$

Now $\hat{\mathbf{k}}$ can be computed by applying the RHR to $\hat{\mathbf{i}}$ and $\hat{\mathbf{j}}$. This results in Equation (5.55).

$$\hat{\mathbf{k}} = \hat{\mathbf{i}} \times \hat{\mathbf{j}} \tag{5.55}$$

This leaves us with the unit vectors in the body frame defined as a function of the state in the inertial frame.

DEFINING THE TRANSFORMATION MATRIX COEFFICIENTS

Now, if the acceleration in the x-direction in the body frame is multiplied with $\hat{\mathbf{i}}$ this a_{x_B} is redefined in the inertial frame by providing a contribution in the x-, y- and z-direction. A similar multiplication can be done for a_{y_B} and $\hat{\mathbf{j}}$, and a_{z_B} and $\hat{\mathbf{k}}$. In matrix form this can be represented by Equation (5.56).

$$[\hat{\mathbf{i}} \quad \hat{\mathbf{j}} \quad \hat{\mathbf{k}}] \cdot \begin{pmatrix} a_{x_B} \\ a_{y_B} \\ a_{z_B} \end{pmatrix} \tag{5.56}$$

Comparing Equation (5.56) to the expression for the transformation matrix provided by Equation (5.51) it can be seen that the coefficients can now be expressed as Equation (5.57).

$$\begin{pmatrix} C_1 \\ C_2 \\ C_3 \end{pmatrix} = \hat{\mathbf{i}} \qquad \begin{pmatrix} C_4 \\ C_5 \\ C_6 \end{pmatrix} = \hat{\mathbf{j}} \qquad \begin{pmatrix} C_7 \\ C_8 \\ C_9 \end{pmatrix} = \hat{\mathbf{k}} \tag{5.57}$$

Given these definitions, the corresponding auxiliary functions can now be defined.

AUXILIARY FUNCTIONS FOR THE SECOND CASE

With the transformation matrix now defined, Equation (5.14) can be written as Equation (5.58).

$$\begin{pmatrix} a_x \\ a_y \\ a_z \end{pmatrix} = \begin{pmatrix} -\mu_M \frac{x}{r^3} \\ -\mu_M \frac{y}{r^3} \\ -\mu_M \frac{z}{r^3} \end{pmatrix} + [\hat{\mathbf{i}} \quad \hat{\mathbf{j}} \quad \hat{\mathbf{k}}] \cdot \left[\begin{pmatrix} -\frac{D}{m_{MAV}} \\ 0 \\ 0 \end{pmatrix} + \ldots\ldots \Big|_{\mathbf{B}} \mathbb{T}_{\mathbf{z}}(-\psi_T)\, \mathbb{T}_{\mathbf{y}}(-\epsilon_T) \Big|_{\mathbf{P}} \begin{pmatrix} \frac{T}{m_{MAV}} \\ 0 \\ 0 \end{pmatrix} \right] \tag{5.58}$$

In this case, all the auxiliary functions involving the rotation angles and the transformations ($w_{4,5} - w_{4,8}$, $w_{4,13} - w_{4,25}$, $w_{4,40} - w_{4,52}$, $w_{5,2} - w_{5,6}$ and $w_{6,2} - w_{6,7}$) can be discarded and instead new auxiliary functions have to be defined for the new transformation matrix. Please note that the discarded numbers will be reused but now with different function definitions for the second case!

The first auxiliary functions can found by rewriting Equation (5.53) as a function of the non-discarded auxiliary functions described in Section 5.2.1 and are shown in Equation (5.59).

$$w_{4,13} = C_1 = \frac{V_x + \Omega_M y}{V_G} = \frac{w_{4,9}}{w_{4,12}} \qquad w_{4,14} = C_2 = \frac{V_y - \Omega_M x}{V_G} = \frac{w_{4,10}}{w_{4,12}} \qquad w_{4,15} = C_3 = \frac{V_z}{V_G} = \frac{x_6}{w_{4,12}} \tag{5.59}$$

In case of the auxiliary functions for $\hat{\mathbf{j}}$ first the cross product has to be computed and the norm of that cross product. This is done in Equation (5.60).

$$\begin{aligned} w_{4,16} &= x_6 x_2 - w_{4,10} x_3 \\ w_{4,17} &= w_{4,9} x_3 - x_6 x_1 \\ w_{4,18} &= w_{4,10} x_1 - w_{4,9} x_2 \end{aligned} \qquad \begin{aligned} w_{4,19} &= w_{4,16}^2 + w_{4,17}^2 + w_{4,18}^2 \\ w_{4,20} &= \sqrt{w_{4,19}} \end{aligned} \qquad \begin{aligned} w_{4,21} &= C_4 = \frac{w_{4,16}}{w_{4,20}} \\ w_{4,22} &= C_5 = \frac{w_{4,17}}{w_{4,20}} \\ w_{4,23} &= C_6 = \frac{w_{4,18}}{w_{4,20}} \end{aligned} \tag{5.60}$$

These auxiliary functions can now be used to describe the $\hat{\mathbf{k}}$ functions as shown by Equation (5.61). In this case, the last auxiliary functions was named 40 because 26 is already taken up by an auxiliary function for the thrust acceleration.

$$
\begin{aligned}
w_{4,24} &= C_7 = C_2C_6 - C_3C_5 = w_{4,14}w_{4,23} - w_{4,15}w_{4,22} \\
w_{4,25} &= C_8 = C_3C_4 - C_1C_6 = w_{4,15}w_{4,21} - w_{4,13}w_{4,23} \\
w_{4,40} &= C_9 = C_1C_5 - C_2C_4 = w_{4,13}w_{4,22} - w_{4,14}w_{4,21}
\end{aligned} \tag{5.61}
$$

Then combining everything results in the expression presented in Equation (5.62) for u_4, u_5 and u_6.

$$
\begin{aligned}
u_4 &= w_{4,39} + w_{4,36}w_{4,13} + w_{4,37}w_{4,21} - w_{4,38}w_{4,24} \\
u_5 &= w_{5,1} + w_{4,36}w_{4,14} + w_{4,37}w_{4,22} - w_{4,38}w_{4,25} \\
u_6 &= w_{6,1} + w_{4,36}w_{4,15} + w_{4,37}w_{4,23} - w_{4,38}w_{4,40}
\end{aligned} \tag{5.62}
$$

5.2.3. SPHERICAL EQUATIONS

For the Spherical case, the initial conditions are readily provided. The position comes directly from the chosen launch site and the velocity comes from the initial MAV conditions. These are all defined in the rotating frame as depicted in Figure 5.4 and Equation (5.63).

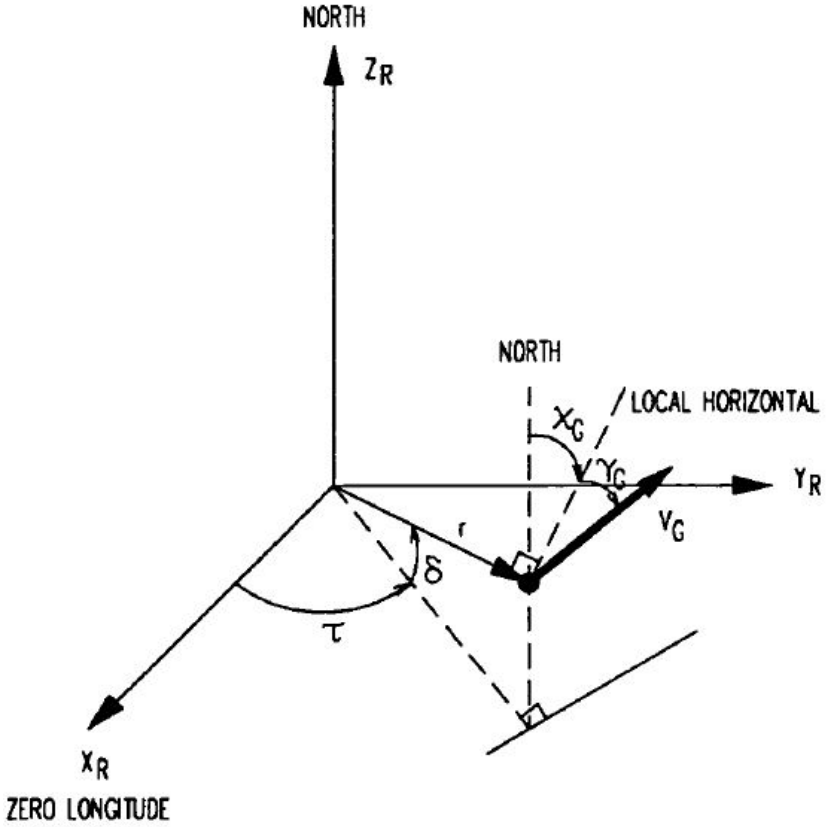


Figure 5.4: Definition of the Spherical coordinates (Mooij, 1994).

$$
\mathbf{R} = \begin{pmatrix} r \\ \delta \\ \tau \end{pmatrix} \qquad \mathbf{V} = \begin{pmatrix} V_G \\ \gamma_G \\ \chi_G \end{pmatrix} \qquad m_{MAV} \tag{5.63}
$$

The different variables are now defined as shown by Equation (5.64) and the derivatives by Equation (5.65). The numbers that were given to the variables are a result of the early derivations.

$$\begin{aligned} x_{16} &= r = h + R_{MOLA} & x_{15} &= V_G \\ x_{12} &= \delta & x_{14} &= \gamma_G & x_7 &= m_{MAV} \\ x_{11} &= \tau & x_{13} &= \chi_G \end{aligned} \tag{5.64}$$

$$\begin{aligned} x'_{16} &= \dot{r} & x'_{15} &= \dot{V}_G \\ x'_{12} &= \dot{\delta} & x'_{14} &= \dot{\gamma}_G & x'_7 &= \dot{m}_{MAV} = -\frac{T}{g_0 I_{sp}} \\ x'_{11} &= \dot{\tau} & x'_{13} &= \dot{\chi}_G \end{aligned} \tag{5.65}$$

The derivatives given in Equation (5.65) are already provided by Mooij (1994) in the rotating frame and include the rotational effects. A simplified form can be seen in Equations (5.66) and (5.67). This is why it was decided to perform the TSI for the Spherical case in the rotating frame. At the end of the integration the variables can then be transformed to the inertial frame for comparison.

$$x'_{16} = \dot{r} = V_G s\gamma_G \qquad x'_{12} = \dot{\delta} = \frac{V_G c\chi_G c\gamma_G}{r} \qquad x'_{11} = \dot{\tau} = \frac{V_G s\chi_G c\gamma_G}{rc\delta} \tag{5.66}$$

$$\begin{aligned} x'_{15} &= \dot{V}_G = \Omega_M^2 rc\delta\left(s\gamma_G c\delta - c\gamma_G s\delta c\chi_G\right) + \frac{Tc\psi_T c\epsilon_T}{m} - \frac{D}{m} - \frac{\mu_M}{r^2} s\gamma_G \\ x'_{14} &= \dot{\gamma}_G = 2\Omega_M c\delta s\chi_G + \frac{V_G}{r} c\gamma_G + \frac{\Omega_M^2 rc\delta}{V_G}\left(c\delta c\gamma_G + s\gamma_G s\delta c\chi_G\right) + \frac{Ts\epsilon_T}{V_G m} - \frac{\mu_M}{V_G r^2} c\gamma_G \\ x'_{13} &= \dot{\chi}_G = 2\Omega_M\left(s\delta - \frac{c\delta s\gamma_G c\chi_G}{c\gamma_G}\right) + \frac{V_G}{r} c\gamma_G \frac{s\delta}{c\delta} s\chi_G + \frac{\Omega_M^2 rc\delta s\delta s\chi_G}{V_G c\gamma_G} + \frac{Ts\psi_T c\epsilon_T}{V_G mc\gamma_G} \end{aligned} \tag{5.67}$$

The last term in the first two expressions of Equation (5.67) correspond to the acceleration contribution of the gravity. The first expression has a drag component and all three have a thrust component similar to the Cartesian cases. The rest of the terms come from the rotational effects. In this case it was decided to define the drag as an auxiliary equation x_{27}, which will be discussed in more detail later in this section.

AUXILIARY FUNCTIONS FOR THE SPHERICAL CASE

Before the auxiliary functions for Equations (5.66) and (5.66) can be written, some standard auxiliary functions have to be defined first. These include δ, γ_G and χ_G. The numbering is based on early derivations and do not correspond to the definitions provided for the Cartesian cases! The angle auxiliary functions are defined in Equation (5.68).

$$\begin{aligned} w_{4,4} &= s\delta = sx_{12} & w_{4,7} &= s\gamma_G = sx_{14} \\ w_{4,5} &= c\chi_G = cx_{13} & w_{4,8} &= c\gamma_G = cx_{14} \\ w_{4,6} &= c\delta = cx_{12} & w_{4,9} &= s\chi_G = sx_{13} \end{aligned} \tag{5.68}$$

Some expression for the thrust are also required. These are similar to the equations defined in Section 5.2.1 but are repeated in Equation (5.69) for convenience.

$$\begin{aligned} w_{4,26} &= \cos\psi_T & w_{4,28} &= \sin\psi_T & w_{4,32} &= \frac{1}{x_7} \\ w_{4,27} &= \cos\epsilon_T & w_{4,29} &= \sin\epsilon_T \end{aligned} \tag{5.69}$$

Now the auxiliary functions can be written for $x'_{11} = u_{11}$ till $x'_{16} = u_{16}$ and are shown in Equations (5.70) to (5.75) respectively.

$$\begin{aligned} w_{11,0} &= V_G c\gamma_G = x_{15} w_{4,8} & w_{11,2} &= w_{11,1} s\chi_G = w_{11,1} w_{4,9} & u_{11} &= w_{11,3} \\ w_{11,1} &= \frac{w_{11,0}}{r} = \frac{w_{11,0}}{x_{16}} & w_{11,3} &= \frac{w_{11,2}}{c\delta} = \frac{w_{11,2}}{w_{4,6}} \end{aligned} \tag{5.70}$$

$$w_{12,1} = w_{11,1} c\chi_G = w_{11,1} w_{4,5} \qquad u_{12} = w_{12,1} \tag{5.71}$$

$$\begin{aligned}
w_{13,0} &= s\psi_T c\epsilon_T = w_{4,28} w_{4,27} \\
w_{13,1} &= s\gamma_G c\chi_G = w_{4,7} w_{4,5} \\
w_{13,2} &= \Omega_M^2 r c\delta = \Omega_M^2 x_{16} w_{4,6} \\
w_{13,3} &= w_{13,2} s\delta = w_{13,2} w_{4,4} \\
w_{13,4} &= \frac{T w_{13,0}}{x_7} = T w_{13,0} w_{4,32} \\
w_{13,5} &= w_{13,3} s\chi_G + w_{13,4} = w_{13,3} w_{4,9} + w_{13,4} \\
w_{13,6} &= \frac{w_{13,5}}{V_G} = \frac{w_{13,5}}{x_{15}} \\
w_{13,7} &= -2\Omega_M c\delta w_{13,1} + w_{13,6} = -2\Omega_M w_{4,6} w_{13,1} + w_{13,6} \qquad u_{13} = w_{13,9} \\
w_{13,8} &= \frac{w_{13,7}}{c\gamma_G} = \frac{w_{13,7}}{w_{4,8}} \\
w_{13,9} &= \left(2\Omega_M + w_{11,3}\right) s\delta + w_{13,8} = \left(2\Omega_M + w_{11,3}\right) w_{4,4} + w_{13,8}
\end{aligned} \tag{5.72}$$

$$\begin{aligned}
w_{14,0} &= \frac{T s\epsilon_T}{x_7} = T w_{4,29} w_{4,32} \\
w_{14,1} &= c\delta c\gamma_G + w_{13,1} s\delta = w_{4,6} w_{4,8} + w_{13,1} w_{4,4} \\
w_{14,2} &= -\mu_M c\gamma_G = -\mu_M w_{4,8} \\
w_{14,3} &= r^2 = x_{16}^2 \\
w_{14,4} &= \frac{w_{14,2}}{w_{14,3}} \\
w_{14,5} &= w_{14,4} + w_{13,2} w_{14,1} + w_{14,0} \\
w_{14,6} &= \frac{w_{14,5}}{V_G} = \frac{w_{14,5}}{x_{15}} \\
w_{14,7} &= 2\Omega_M c\delta s\chi_G + w_{11,1} + w_{14,6} = 2\Omega_M w_{4,6} w_{4,9} + w_{11,1} + w_{14,6} \\
u_{14} &= w_{14,7}
\end{aligned} \tag{5.73}$$

$$\begin{aligned}
w_{15,0} &= T c\psi_T c\epsilon_T - D = T w_{4,26} w_{4,27} - x_{27} \\
w_{15,1} &= \frac{w_{15,0}}{m} = \frac{w_{15,0}}{x_7} \\
w_{15,2} &= -\mu_M s\gamma_G = -\mu_M w_{4,7} \\
w_{15,3} &= \frac{w_{15,2}}{w_{14,3}} \\
w_{15,4} &= c\gamma_G c\chi_G = w_{4,8} w_{4,5} \\
w_{15,5} &= s\gamma_G c\delta - w_{15,4} s\delta = w_{4,7} w_{4,6} - w_{15,4} w_{4,4} \\
w_{15,6} &= w_{13,2} w_{15,5} + w_{15,1} + w_{15,3} \\
u_{15} &= w_{15,6}
\end{aligned} \tag{5.74}$$

$$w_{16,1} = V_G s\gamma_G = x_{15} w_{4,7} \qquad u_{16} = w_{16,1} \tag{5.75}$$

DRAG ACCELERATION AS AUXILIARY VARIABLE

The drag acceleration for the Spherical case is similar to the Cartesian case, except this time it was chosen to use auxiliary variables to describe the drag. This means that auxiliary equations have to be defined as is shown in Equation (5.76). Also, this means that auxiliary derivatives are required for each of these equations. Equation (5.77) shows the different derivatives corresponding to the equations.

$$\begin{aligned}
x_{27} &= D = \frac{1}{2}\rho V_G^2 S C_D = \frac{1}{2} S x_{28} x_{15}^2 x_{29} \\
x_{28} &= \rho = e^{x_{30}} \\
x_{29} &= C_D \\
x_{30} &= P_{\rho 10} x_{31}^{10} + P_{\rho 9} x_{31}^9 + P_{\rho 8} x_{31}^8 + P_{\rho 7} x_{31}^7 + P_{\rho 6} x_{31}^6 + P_{\rho 5} x_{31}^5 + P_{\rho 4} x_{31}^4 + P_{\rho 3} x_{31}^3 + P_{\rho 2} x_{31}^2 + P_{\rho 1} x_{31} + P_{\rho 0} \\
x_{31} &= h = r - R_{MOLA} = x_{16} - R_{MOLA}
\end{aligned} \tag{5.76}$$

$$\begin{aligned}
x'_{27} &= \frac{1}{2} S x_{15} \left(x_{15} \left(x_{29} x'_{28} + x_{28} x'_{29} \right) + 2 x_{28} x_{29} x'_{15} \right) \\
x'_{28} &= x'_{30} x_{28} \\
x'_{30} &= x'_{31} \left(10 P_{\rho 10} x_{31}^9 + 9 P_{\rho 9} x_{31}^8 + 8 P_{\rho 8} x_{31}^7 + 7 P_{\rho 7} x_{31}^6 + 6 P_{\rho 6} x_{31}^5 + \ldots \right. \\
&\quad \left. \ldots + 5 P_{\rho 5} x_{31}^4 + 4 P_{\rho 4} x_{31}^3 + 3 P_{\rho 3} x_{31}^2 + 2 P_{\rho 2} x_{31} + P_{\rho 1} \right) \\
x'_{31} &= x'_{16}
\end{aligned} \tag{5.77}$$

The auxiliary derivatives described in Equation (5.77) can now be used to define the auxiliary functions. These are described in Equations (5.78) to (5.80) for u_{27}, u_{28} and u_{30} respectively. In this case $u_{31} = u_{16}$.

$$\begin{aligned}
w_{27,1} &= C_D\dot{\rho} = x_{29}u_{28} & w_{27,4} &= V_G\left(w_{27,1} + w_{27,2}\right) = x_{15}\left(w_{27,1} + w_{27,2}\right) \\
w_{27,2} &= \rho\dot{C}_D = x_{28}u_{29} & w_{27,5} &= w_{27,3}\dot{V}_G = w_{27,3}u_{15} \qquad u_{27} = \frac{1}{2}Sw_{27,6} \\
w_{27,3} &= \rho C_D = x_{28}x_{29} & w_{27,6} &= V_G\left(w_{27,4} + w_{27,5}\right) = x_{15}\left(w_{27,4} + w_{27,5}\right)
\end{aligned} \tag{5.78}$$

$$w_{28,1} = u_{30}\rho = u_{30}x_{28} \qquad u_{28} = w_{28,1} \tag{5.79}$$

$$\begin{aligned}
w_{30,1} &= h^9 = x_{31}^9 & w_{30,6} &= h^4 = x_{31}^4 \\
w_{30,2} &= h^8 = x_{31}^8 & w_{30,7} &= h^3 = x_{31}^3 \\
w_{30,3} &= h^7 = x_{31}^7 & w_{30,8} &= h^2 = x_{31}^2 \\
w_{30,4} &= h^6 = x_{31}^6 & w_{30,9} &= u_{31}\left(10P_{\rho 10}w_{30,1} + \cdots + 3P_{\rho 3}w_{30,8} + 2P_{\rho 2}x_{31} + P_{\rho 1}\right) \\
w_{30,5} &= h^5 = x_{31}^5 & u_{30} &= w_{30,9}
\end{aligned} \tag{5.80}$$

Again, three additional auxiliary equations are required for the $C_D - M$ relation (see Figure 2.4) and are presented in Equation (5.81).

$$\begin{aligned}
x_{32} &= M = \frac{V_G}{a} = \frac{x_{15}}{x_{33}} \\
x_{33} &= a = \sqrt{\gamma_a R_a^* T_a} = \sqrt{\gamma_a R_a^* x_{34}} \quad \text{where} \quad R_a^* = \frac{R_a}{M_a} \\
x_{34} &= T_a
\end{aligned} \tag{5.81}$$

In this case the conditional relations shown in Equation (5.82) are not used directly in the recurrence relations, but only serve as initial conditions. Again, the $P_{C_D number,section}$ are the polynomial fit coefficients as provided in Table 3.5.

$$x_{29} = C_D = \begin{cases}
x_{29,1} = 0.2, & \text{for } 0 \le x_{32} < 0.5 \\
x_{29,2} = P_{C_D 1,2}x_{32} + P_{C_D 0,2}, & \text{for } 0.5 \le x_{32} < 1 \\
x_{29,3} = P_{C_D 1,3}x_{32} + P_{C_D 0,3}, & \text{for } 1 \le x_{32} < 1.3 \\
x_{29,4} = P_{C_D 1,4}x_{32} + P_{C_D 0,4}, & \text{for } 1.3 \le x_{32} < 2.5 \\
x_{29,5} = P_{C_D 1,5}x_{32} + P_{C_D 0,5}, & \text{for } 2.5 \le x_{32} < 4 \\
x_{29,6} = 0.3, & \text{for } x_{32} \ge 4
\end{cases} \tag{5.82}$$

The corresponding derivatives that will have to be used for the recurrence relations are presented in Equation (5.83).

$$x'_{29} = \begin{cases}
x'_{29,1} = 0, & \text{for } 0 \le x_{32} < 0.5 \\
x'_{29,2} = P_{C_D 1,2}x'_{32}, & \text{for } 0.5 \le x_{32} < 1 \\
x'_{29,3} = P_{C_D 1,3}x'_{32}, & \text{for } 1 \le x_{32} < 1.3 \\
x'_{29,4} = P_{C_D 1,4}x'_{32}, & \text{for } 1.3 \le x_{32} < 2.5 \\
x'_{29,5} = P_{C_D 1,5}x'_{32}, & \text{for } 2.5 \le x_{32} < 4 \\
x'_{29,6} = 0, & \text{for } x_{32} \ge 4
\end{cases} \tag{5.83}$$

The Mach derivative and corresponding speed of sound derivative are then described in Equation (5.84).

$$\begin{aligned}
x'_{32} &= \frac{x_{33}x'_{15} - x_{15}x'_{33}}{x_{33}^2} \\
x'_{33} &= \frac{\gamma_a R_a^*}{2x_{33}}x'_{34}
\end{aligned} \tag{5.84}$$

For the drag coefficient, there are no auxiliary functions required. That is why x' in Equation (5.83) can simply be replaced by u. The Mach number and the speed of sound do require auxiliary functions and are described in Equations (5.85) and (5.86) respectively.

$$\begin{aligned} w_{32,1} &= a\dot{V}_G = x_{33}u_{15} & w_{32,3} &= a^2 = x_{33}^2 \\ w_{32,2} &= V_G\dot{a} = x_{15}u_{33} & w_{32,4} &= \frac{w_{32,1} - w_{32,2}}{w_{32,3}} \end{aligned} \qquad u_{32} = w_{32,4} \tag{5.85}$$

$$w_{33,1} = \frac{\dot{T}_a}{a} = \frac{u_{34}}{x_{33}} \qquad u_{33} = w_{33,1} \tag{5.86}$$

The last relations that still need to be described are the ones for the temperature T_a. Similar to C_D the corresponding auxiliary equations, shown by Equation (5.87) are only used for the initial conditions. Again, $P_{Tnumber,section}$ are the polynomial fit coefficients as provided in Table 3.2.

$$x_{34} = T_a = \begin{cases} x_{34,1} = P_{T1,1}x_{31} + P_{T0,1}, & \text{for } -0.6 \le x_{31} < 5.04 \\ x_{34,2} = P_{T3,2}x_{31}^3 + P_{T2,2}x_{31}^2 + P_{T1,2}x_{31} + P_{T0,2}, & \text{for } 5.04 \le x_{31} < 35.53 \\ x_{34,3} = P_{T6,3}x_{31}^6 + P_{T5,3}x_{31}^5 + P_{T4,3}x_{31}^4 + P_{T3,3}x_{31}^3 + \dots & \\ \qquad \dots + P_{T2,3}x_{31}^2 + P_{T1,3}x_{31} + P_{T0,3}, & \text{for } 35.53 \le x_{31} < 75.07 \\ x_{34,4} = P_{T8,4}x_{31}^8 + P_{T7,4}x_{31}^7 + P_{T6,4}x_{31}^6 + P_{T5,4}x_{31}^5 & \\ \qquad + P_{T4,4}x_{31}^4 + P_{T3,4}x_{31}^3 + P_{T2,4}x_{31}^2 + P_{T1,4}x_{31} + P_{T0,4}, & \text{for } 75.07 \le x_{31} < 170.05 \\ x_{34,5} = 136.5, & \text{for } x_{31} \ge 170.05 \end{cases} \tag{5.87}$$

The corresponding derivatives that are used for the recurrence relations are presented in Equation (5.88).

$$x'_{34} = \begin{cases} x'_{34,1} = P_{T1,1}x'_{31}, & \text{for } -0.6 \le x_{31} < 5.04 \\ x'_{34,2} = \left(3P_{T3,2}x_{31}^2 + 2P_{T2,2}x_{31} + P_{T1,2}\right)x'_{31}, & \text{for } 5.04 \le x_{31} < 35.53 \\ x'_{34,3} = \left(6P_{T6,3}x_{31}^5 + 5P_{T5,3}x_{31}^4 + 4P_{T4,3}x_{31}^3 + \dots\right. & \\ \qquad \left. \dots + 3P_{T3,3}x_{31}^2 + 2P_{T2,3}x_{31} + P_{T1,3}\right)x'_{31}, & \text{for } 35.53 \le x_{31} < 75.07 \\ x'_{34,4} = \left(8P_{T8,4}x_{31}^7 + 7P_{T7,4}x_{31}^6 + 6P_{T6,4}x_{31}^5 + 5P_{T5,4}x_{31}^4 + \dots\right. & \\ \qquad \left. \dots + 4P_{T4,4}x_{31}^3 + 3P_{T3,4}x_{31}^2 + 2P_{T2,4}x_{31} + P_{T1,4}\right)x'_{31}, & \text{for } 75.07 \le x_{31} < 170.05 \\ x'_{34,5} = 0, & \text{for } x_{31} \ge 170.05 \end{cases} \tag{5.88}$$

All the power relations in Equation (5.88) were already described in auxiliary functions in Equation (5.80). Therefore, depending on the section, the equations presented in Equation (5.88) can be rewritten by simply replacing x' by u and the auxiliary functions. This is shown in Equation (5.89).

$$u_{34} = \begin{cases} u_{34,1} = P_{T1,1}u_{31}, & \text{for } -0.6 \le x_{31} < 5.04 \\ u_{34,2} = \left(3P_{T3,2}w_{30,2} + 2P_{T2,2}x_{31}\right)u_{31} + P_{T1,2}u_{31}, & \text{for } 5.04 \le x_{31} < 35.53 \\ u_{34,3} = \left(6P_{T6,3}w_{30,5} + 5P_{T5,3}w_{30,4} + 4P_{T4,3}w_{30,3} + \dots\right. & \\ \qquad \left. \dots + 3P_{T3,3}w_{30,2} + 2P_{T2,3}x_{31}\right)u_{31} + P_{T1,3}u_{31}, & \text{for } 35.53 \le x_{31} < 75.07 \\ u_{34,4} = \left(8P_{T8,4}w_{30,7} + 7P_{T7,4}w_{30,6} + 6P_{T6,4}w_{30,5} + 5P_{T5,4}w_{30,4} + \dots\right. & \\ \qquad \left. \dots + 4P_{T4,4}w_{30,3} + 3P_{T3,4}w_{30,2} + 2P_{T2,4}x_{31}\right)u_{31} + P_{T1,4}u_{31}, & \text{for } 75.07 \le x_{31} < 170.05 \\ u_{34,5} = 0, & \text{for } x_{31} \ge 170.05 \end{cases} \tag{5.89}$$

Now all the information that is required to compute the recurrence relation for the drag is available. Please note that there is a certain order in which these equations have to be computed, since they depend on each other. Table 5.1 shows this order.

Table 5.1: Order of auxiliary equations and derivatives computations

Auxiliary equations				Auxiliary derivatives			
Order	$\mathbf{x_n}$	**Order**	$\mathbf{x_n}$	**Order**	$\mathbf{u_n}$	**Order**	$\mathbf{u_n}$
0	$x_7, x_{11}, x_{12}, x_{13}, x_{14}, x_{15}, x_{16}$	4	x_{32}	7	$u_7, u_{11}, u_{12}, u_{16}$	11	u_{32}
1	x_{31}	5	x_{29}	8	u_{31}	12	u_{29}
2	x_{30}, x_{34}	6	x_{27}	9	u_{30}, u_{34}	13	u_{27}
3	x_{33}, x_{28}			10	u_{28}, u_{33}	14	u_{13}, u_{14}, u_{15}

5.2.4. RECURRENCE RELATIONS

Now that all the auxiliary functions are known, they can be used to determine the required recurrence relations. In order to write these concisely and following the same convention as used by Scott and Martini (2008), a number of extra parameters will have to be introduced. The Taylor series coefficients can be written as described by Equation (5.90). Where n is the variable number and k is the order of the derivative.

$$\frac{x_n^{(k)}}{k!} \triangleq X_n(k) \quad \text{where} \quad k \geq 1 \tag{5.90}$$

A similar expression is defined for u_n and w_n as well as the place-holder functions f_n and g_n which are all shown in Equation (5.91).

$$U_n(k) \triangleq \frac{u_n^{(k)}}{k!} \qquad W_n(k) \triangleq \frac{w_n^{(k)}}{k!} \qquad F_n(k) \triangleq \frac{f_n^{(k)}}{k!} \qquad G_n(k) \triangleq \frac{g_n^{(k)}}{k!} \tag{5.91}$$

Then remembering that $u_n = x_n'$ and using Equation (5.90) a relation can be described between X_n and U_n as shown in Equation (5.92) (Scott and Martini, 2008).

$$\begin{aligned} u_n^{(k-1)} = x_n^{(k)} \quad &\Rightarrow \quad \frac{u_n^{(k-1)}}{(k-1)!} = \frac{x_n^{(k)}}{(k-1)!} \quad \Rightarrow \\ U_n(k-1) = kX_n(k) \quad &\Rightarrow \quad X_n(k) = \frac{U_n(k-1)}{k} \end{aligned} \tag{5.92}$$

Using these definitions, the auxiliary functions can be written into reduced auxiliary functions $W(k)$ for the three cases. The equations stay the same, but the lower-case letters are replaced by their upper-case counterparts. A complete list of the reduced auxiliary functions (starting at $k = 2$, or the second derivative u') for the three cases is presented in Appendix D. As was mentioned before, all the auxiliary functions have been defined such that they incorporate one of the following operations: multiplication, division, power, exponential or trigonometric. This has been done because recurrence relations exist for these simple operations as provided by Jorba and Zou (2005). These recurrence relations are written using the definitions from Equation (5.91) and are described in Equations (5.93) to (5.99).

$$\text{for} \quad f_n \pm g_n \quad \Rightarrow \quad W_{n,\pm}(k) = F_n(k) \pm G_n(k) \tag{5.93}$$

$$\text{for} \quad f_n g_n \quad \Rightarrow \quad W_{n,mult}(k) = \sum_{j=0}^{k} F_n(j)\, G_n(k-j) \tag{5.94}$$

$$\text{for} \quad \frac{f_n}{g_n} \quad \Rightarrow \quad W_{n,div}(k) = \frac{1}{G_n(0)} \left[F_n(k) - \sum_{j=1}^{k} G_n(j)\, W_{n,div}(k-j) \right] \tag{5.95}$$

$$\text{for} \quad f_n^{\alpha} \quad \Rightarrow \quad W_{n,pow}(k) = \frac{1}{kF_n(0)} \sum_{j=0}^{k-1} \left[k\alpha - j(\alpha+1) \right] F_n(k-j)\, W_{n,pow}(j) \tag{5.96}$$

$$\text{for} \quad e^{f_n} \quad \Rightarrow \quad W_{n,exp}(k) = \frac{1}{k} \sum_{j=0}^{k-1} (k-j)\, W_{n,exp}(j)\, F_n(k-j) \tag{5.97}$$

$$\text{for} \quad \cos f_n \quad \Rightarrow \quad W_{n,cos}(k) = -\frac{1}{k}\sum_{j=1}^{k} j\, W_{n,sin}(k-j)\, F_n(j) \tag{5.98}$$

$$\text{for} \quad \sin f_n \quad \Rightarrow \quad W_{n,sin}(k) = \frac{1}{k}\sum_{j=1}^{k} j\, W_{n,cos}(k-j)\, F_n(j) \tag{5.99}$$

Checking the presented equations it can be seen that Equations (5.98) and (5.99) are interdependent. This means that whenever a cosine recurrence relation has to be computed, the same recurrence relation for sine (with at least order $k-1$) has to be computed at the same time (and vice versa). All these equations have been derived by Jorba and Zou (2005) using the general Leibniz rule for the k^{th} derivative of a multiplication as portrayed in Equation (5.100).

$$(f_n g_n)^{(k)} = \sum_{j=0}^{k} \binom{k}{j} f_n^{(j)} g_n^{(k-j)} \tag{5.100}$$

Combining Equation (5.100) and the definition of the reduced derivative for the auxiliary function (see Equation (5.91)) results in Equation (5.101). This equation, when rewritten to include the reduced derivatives for f_n and g_n, results in Equation (5.94).

$$W_{n,mult}(k) = \frac{1}{k!}\sum_{j=0}^{k} \binom{k}{j} f_n^{(j)} g_n^{(k-j)} \tag{5.101}$$

With all the recurrence relations now known, and using the definition of Equation (5.92) the updated state can be described using the Taylor series expansion as described in Equation (5.102).

$$x_n(t+h) = \sum_{k=0}^{K} \frac{x_n^{(k)}(t)}{k!} h^k + T_{n,K} = \sum_{k=0}^{K} X_n(k)\, h^k + T_{n,K} \quad \text{with } n = 1,\ldots,7 \tag{5.102}$$

Here K is the order of the series to which it has to be evaluated and $T_{n,K}$ is the truncation error. Please note that all the Taylor series coefficients computed for the auxiliary equations are only needed to determine the Taylor series coefficients of the state variables (which is why they are auxiliary).

6

PROGRAM SIMULATION TOOL

To perform the analysis associated with this thesis, a simulation program has been written. This tool is comprised of both existing (Section 6.1) and newly developed software (Section 6.2). 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 using RKF and TSI. This tool is written such that the performance of both integrators can be compared. The final workings of the complete tool is described in Section 6.3.

6.1. 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 own 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 Mars-GRAM) by the mission design section at JPL.

6.1.1. 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 6.1.2 and 6.1.3 respectively. Figure 6.1 shows Tudat with the different libraries that are used within the Tudat Bundle including the core functions (Tudat Core). In this thesis, the Tudat libraries are used for all standard mathematical and astrodynamic operations.

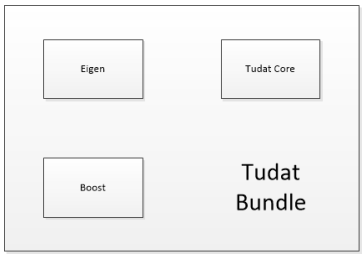


Figure 6.1: Tudat structure

6.1.2. 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.

6.1.3. 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).

6.1.4. 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) [3]. The model is based on NASA Ames Mars General Circulation Model (for altitudes between 0-80 km) and Mars Thermospheric General Circulation model (for altitudes 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 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 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.

6.2. 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.

[1] More documentation on Eigen can be found on eigen.tuxfamily.org/dox/ [Accessed 8 March 2016]
[2] More documentation on Boost can be found on http://www.boost.org/ [Accessed 8 March 2016]
[3] NASA website: https://see.msfc.nasa.gov/model-Marsgram [Accessed 9 March 2016]

6.2.1. RKF PROPAGATOR

The RKF (or traditional) propagator architecture is described in Figure 6.2. It starts with the current state, which is then passed on to the state derivative function. The state derivative function is used by the 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 RKF methods 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.

All blocks represent a different action. These actions might be performed in classes, header files and/or source files. More information on the classification of the different blocks can be found in Appendix C.

6.2.2. TSI PROPAGATOR

The TSI propagator has a significantly different architecture compared to the traditional propagator as can be seen in Figure 6.3. 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 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 the 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.

6.3. COMPLETED SIMULATION TOOL

The final simulation tool combined the integration software in such a way that a complete trajectory propagation could be simulated. It consists of different phases: initialisation, initial propagation, integrator burn phase, integrator coast phase, final circularisation and comparison. A brief description of each phase is provided in this section. It should be noted that in the actual program, the TSI propagation (burn and coast phase) supersedes the RKF propagation. Also, three different RKF methods can be chosen: Runge-Kutta-Fehlberg 4^{th} (5^{th}) order (RKF45), RKF78 and DOPRIN87.

6.3.1. INITIALISATION

During the initialisation phase, all the input values are loaded into the program. This is done by calling the input .txt files and assigning all the proper values to the corresponding parameters. Then, the Mars celestial body class and the MAV class are initialised, providing the Mars atmospheric and planetary data and the MAV system characteristics respectively. At this point, any values that differ from the pre-set default values are updated in both classes. The next step is to convert the initial spherical state into the initial Cartesian state and both states are saved in the respective output files for both TSI and RKF. Finally, the StateAndTime class is initialised for the TSI.

6.3.2. INITIAL PROPAGATION

As will be described in Chapter 7, during the verification process it was found that TSI had trouble dealing with initial conditions close to its singular values during the initial second(s), both for velocity and flight-path angle. This is why it was decided to slightly shift the initial conditions at which the actual integration

for TSI would start. Because RKF did not have similar issues with the first second(s), it was chosen to use an RKF method to propagate the first second. For this initial integration, the default integrator is RKF78 with a tolerance of $1 \cdot 10^{-15}$. The output of this initial propagation is then used as the initial state for both TSI and RKF for the remainder of the integration. This way, both methods can still be compared properly.

6.3.3. INTEGRATOR BURN PHASE

At the start of the burn phase the final state of the initial propagation is taken as the initial state input for the integrators. During the burn phase, both TSI and RKF are initiated and run until the end of the first burn phase occurs. Within this phase, TSI stops at every altitude and Mach section of the atmospheric graphs to make sure that no errors occur in the model. Provided that the atmospheric graphs are piecewise continuous, such stops are not required for RKF. The end of the first burn phase can either be based on a set time or a set altitude. Of course, extra cut-off criteria can be added. During the propagation, all intermediate states are stored in the output files for both methods.

6.3.4. INTEGRATOR COAST PHASE

After the first burn phase, the MAV enters a coasting phase. This is simulated by setting the thrust equal to zero. For TSI this can simply be done by setting the thrust in the MAV class zero within the propagation do-loop. This means that when the set burn-out condition is met, the class is updated, and the loop is continued. However, it is not possible to do this within one loop for the RKF, because of the manner in which RKF is initialised. In Tudat it is described that in order to use the build in RKF functions, a separate state derivative class has to be build first. This class is initialised using the initial state classes, and therefore does not change when updating the initial state classes in the loop directly. This means that when the thrust changes to zero, the state derivative class has to be re-initialised, creating the need for a second separate do-loop. This also means that the integrator class has to be re-initialised as well, which means that a separate integrator has to be defined (one with a different name from the first one). Both the TSI and RKF coasting phases are terminated whenever the desired altitude is reached, or after a certain limit time. Again, all the intermediate states are stored in the output files during the coasting phase.

6.3.5. FINAL CIRCULARISATION

Once the final condition is met, both the RKF and the TSI final state are used to determine the required final circularisation burn in order to reach the desired final orbit. For this computation, the states are first converted into Kepler elements using the Tudat library. Then the required ΔV is computed using the methods described by Wakker (2010). In this case, an instantaneous burn to change the inclination, flight-path angle and orbital velocity to match the desired orbit is assumed. From the ΔV for both methods, it can be determined, using the Tsiolkovsky rocket equation what the required propellant mass is. This then results in a final mass estimate for both the RKF and the TSI propagations.

6.3.6. COMPARISON

Finally, the end states of both TSI and RKF are compared to each other. The program provides the numeric difference between the position, velocity and mass as well as the quotient between those values. This quotient, or difference fraction, is the TSI state divided by the RKF state and the numeric difference is the TSI state minus the RKF state. These values were used in the verification and validation phase and are used during the thesis analysis as well to show the behaviour of TSI compared to RKF.

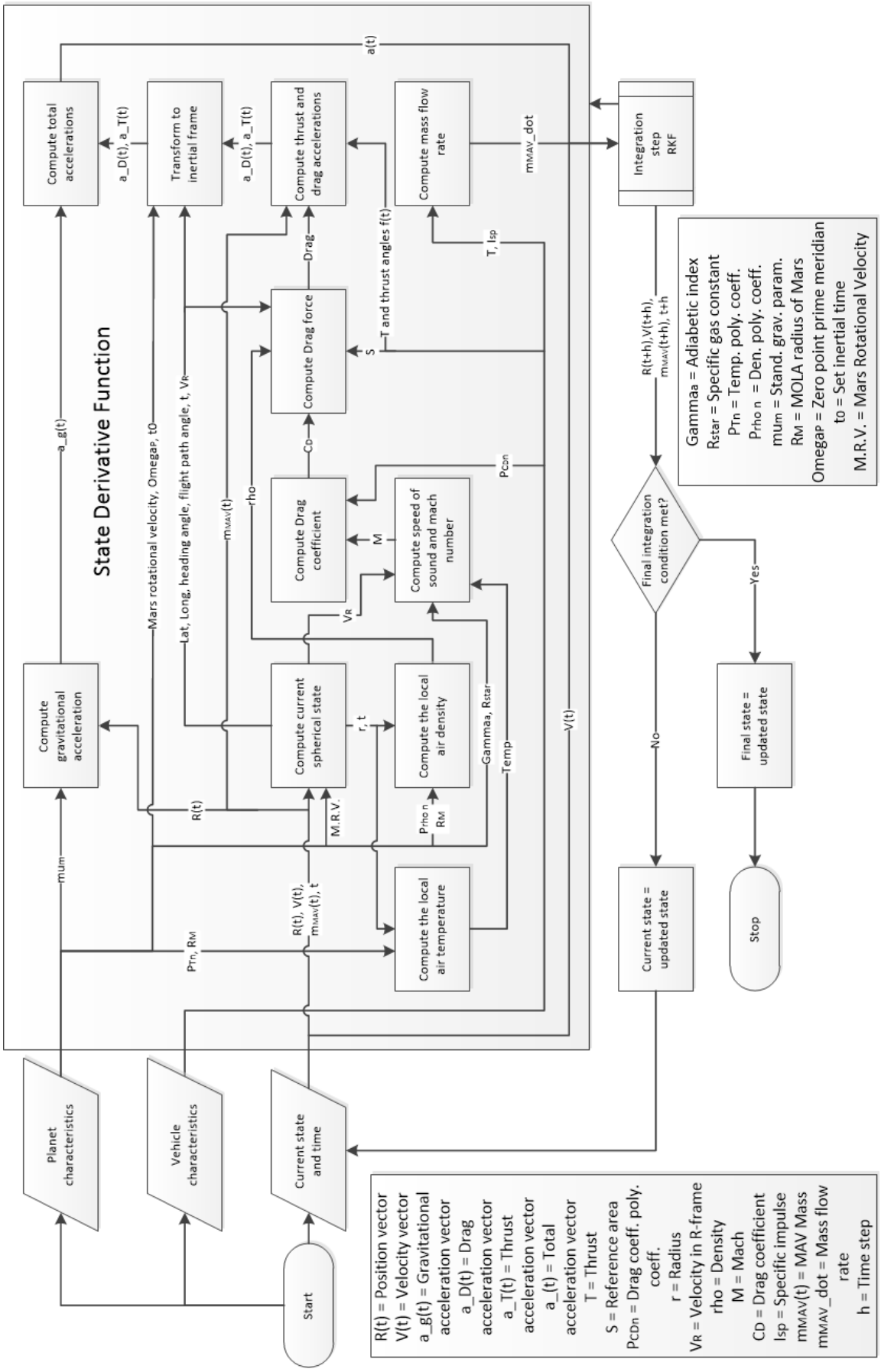


Figure 6.2: RKF interface architecture

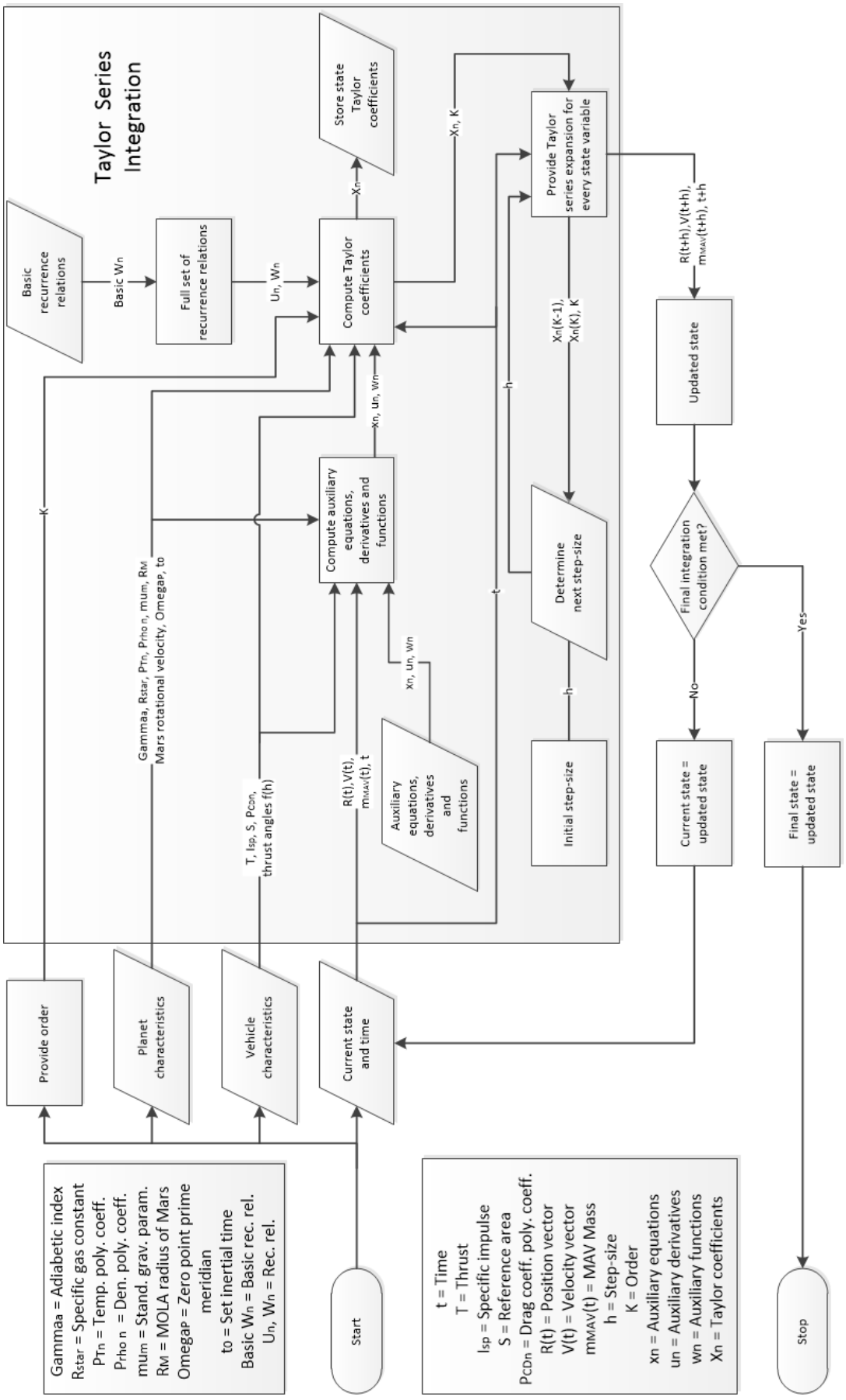


Figure 6.3: TSI architecture

7

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 tests 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 7.1.

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

7.1. 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 RKF. 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 7.1.

Table 7.1: 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), the initial time and the initial state. Using the .integrateTo extension the end state at a certain end time can be computed. 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 7.1.

The second script was used to test the variable step-size integrators RKF. 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 (problem specific), 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 7.1 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 simulation 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.

For the actual propagation tool, the RKF method required the state derivative function associated with this particular ascent problem. The state derivative class was constructed using the state derivative functions as described in this thesis in the Cartesian coordinate system based on equations provided by Mooij (1994). The state is used to compute the required transformation angles first and then both the thrust and drag accelerations are transformed to the inertial frame. There they are combined with the gravitational acceleration resulting in the accelerations in the x-, y- and z-direction. This class was tested by itself by inputting certain values which would result in known outcomes. And those simple values by themselves provided the expected results. Later the state derivative function was combined with the RKF integrator in the trajectory propagation tool and verified again as will be described in Section 7.3.

The verification of the state derivative class took approximately one day.

7.2. TAYLOR SERIES INTEGRATION

In this section, the verification and validation of the TSI header and source file is described. This was done for each of the blocks shown in Figure 6.3. For the verification it was important that the running of the different classes and header/source files did not produce any obvious errors. For the validation, the results were often checked against manual calculations and expected behaviour. The verification and validation of TSI was a very lengthy process and required many rewrites of the used equations because of mistakes in the model or mistakes in the derivations. Section 7.3 talks a bit more about the model mistakes. This section has been divided into several subsections which showed similar issues during the verification and validation.

7.2.1. AUXILIARY EQUATIONS, DERIVATIVES AND FUNCTIONS

As shown by Figure 6.3 the first step in the TSI is to compute the auxiliary equations, derivatives and functions. The computations, in theory, are fairly straightforward, however it is really easy to make a mistake. During the initial verification of these different classes the outcomes were checked against simple manual solutions. Thus, provided a certain input, the output was known. This was an easy way to check any coding errors. Since the early versions of the code required a lot of equations, it took a long time to go through all the equations and find all the errors and mistakes. This was initially done with the combination of Cartesian coordinates and spherical transformation angles, however, later on more efficient equations were used for separate Cartesian and spherical cases. This also made it easier to identify mistakes in the equations and understand certain issues found during the validation process.

7.2.2. COMPUTING THE TAYLOR SERIES COEFFICIENTS AND THE UPDATED STATE

Once the auxiliary classes were verified, the Taylor Series coefficients could be computed using the basic recurrence relations and the full set of recurrence relations. For the Taylor Series coefficients it is important that they show a decreasing number series for each of the coefficients. This was the desired outcome that was used for the verification of the full set of recurrence relations. The basic recurrence relations were all checked by hand calculations to determine if they indeed worked the way that they were suppose to. The verified basic recurrence relations were used to verify the full set of recurrence relations. This class consists of the equations mentioned in Appendix D. When writing the full set of recurrence relations, it is really easy to make mistakes and there are a lot of rules that have to be taken into account as mentioned in Chapter 5. Therefore, a lot of time was spend on rewriting these equations in such a way that the outcomes were what was expected. Also, a number of these recurrence relations were checked by manual calculations. An often recurring problem was that the Taylor Series coefficients would show a diverging behaviour where the coef-

ficient would keep increasing to infinity (theoretically). This was often an indication that a writing mistake was made when setting up the full set of recurrence relations. Another easy way to identify mistakes during the validation process was to compute the next state. Should an error occur in the computation of the Taylor Series coefficients, this would usually result in very large increases in the state even if the coefficient itself would be relatively small. Then following the trail back to the largest change in Taylor Series coefficient, often a coding or writing mistake could be identified in the full set of recurrence relations.

7.2.3. NEXT STEP-SIZE

Fortunately, the step-size class was not a very large class and took less time to verify and validate. Provided a set of Taylor Series coefficients, the step-size class could be tested separately from the other files. The verification process was rather straight forward; the class should be able to provide a step-size estimate given the different input values. Also, when similar sets of Taylor Series coefficients were used, the output step-sizes should be rather similar. This was then also part of the validation process. A pure validation process was difficult because there were no reference Taylor series coefficients available, however an estimate for the order of the next step-size could be used. This can however still be improved. Two different step-size determination methods were tested, but in the end it was found that the direct method described by Scott and Martini (2008) and Bergsma (2015) showed more reasonable results and was easier to implement. Also, during the verification process, the iteration method failed to work properly which led to unusable results and the program failing. But the direct method worked very well.

7.3. COMPLETE TRAJECTORY PROPAGATION

For the complete trajectory propagation it was important to have the integrator method as the only difference between the RKF and the TSI propagation. First the initial values were provided and transformed into the proper coordinate system. These would then be fed into the integrator and at the end of the integration, the results would be printed. The verification and validation of the complete trajectory propagation have been split up into several phases.

7.3.1. PHASE 1: RKF PROPAGATION

This was first done for RKF because this method was already validated. Therefore, any mistakes or problems found would lead back to a mistake in the state derivative functions or the model itself. During this phase, a number of problems were identified; certain singularities were found, a limit to the tolerance value was determined for each of the RKF methods (DOPRIN87 does not converge fast enough in many cases for a value of 10^{-15}) and a mistake in the model was found where the wrong velocity was used to determine the drag acceleration. This meant that some of the model equations had to be rewritten for both RKF and TSI.

7.3.2. PHASE 2: BOTH RKF AND TSI

During the second phase of the verification, both RKF and TSI were tested at the same time and the outcomes were compared. Since the RKF method was verified and validated at this point, the outcome of the RKF propagation could be used to verify TSI. This was done using different test cases that focussed on different elements of the code.

Each element was tested separately by activating (or deactivating) the gravity, thrust and drag as well as the rotation of Mars. In each case, simple test cases were used to verify the class, such as: vertical drops from a certain altitude, thrusting in a known direction and flying parabolic trajectories. This was all done in each of the axis directions first and then combining the different axes. First the gravity was tested, then the thrust, then a combination of these two, followed by the drag all in the non-rotating case. The same was then done for the rotating case as well. This was initially done for a zero thrust elevation and thrust azimuth angle, such that the trust was acting directly through the x-axis of the body frame. Also, because the drag by definition works through the x-axis of the body frame, there should not be any accelerations in the y- and z-direction in the body frame. This simplified the problem and meant that some of the equations could be tested without involving all the other equations just yet.

During this phase it was discovered that there was an inconsistency in the manner in which the different accelerations were computed. Basically, the rotational angles were being computed using spherical equations which already incorporated the accelerations and then those angles were then used to transform the Cartesian coordinates and incorporate the accelerations again. Therefore it was decided to rewrite all the equations and actually come up with three different sets: two Cartesian models and one spherical model for

TSI. The associated equations were already described in Chapter 5.

7.3.3. PHASE 3: THREE DIFFERENT MODELS FOR TSI

The third phase consisted of verifying each of these three models in a similar manner as was done during the second phase. During this phase, it was discovered that a starting ground velocity too close to zero would cause singularities in both the Cartesian as well as the spherical model. This is why a small initial ground velocity was introduced, which seemed to work for the second Cartesian case (unit vector transformations) and the spherical case, however, the first Cartesian case (Euler angles) still did not work. At this point it was decided to focus on the other two methods instead because of time constraints. It was also found that the initial flight path angle could not be equal to (or close to) 90 degrees for the spherical case due to singularities in the model. This is why a value of 89 degrees was chosen for the test cases of that particular model. This worked well for gravity and thrust for the spherical and second Cartesian case. But not always.

7.3.4. PHASE 4: ONE FINAL TSI MODEL

As it turned out, setting an initial ground velocity for the TSI models (Cartesian and spherical) was only a temporary solution and did not solve the problem. Apparently the initial second is vital for TSI and when the velocity is too small, the Taylor Series coefficients tend to increase in value. The solution was to off-set the initial conditions to a later point in the trajectory. This off-set was achieved by having the first second of the trajectory be integrated by RKF78 with a set tolerance of 10^{-15} (if the method and the tolerance was not set constant, the results of the integration would be inconsistent). When this was applied to the propagation involving TSI the Taylor Series coefficients dropped to an acceptable number again which could then be used to determine the next state. This worked really well for the spherical case, however the second Cartesian case still had some bugs (either in the model or in the coding). Again, due to time constraints it was decided to solely continue with the spherical model and use that for the rest of the thesis work. This is the reason that an initial propagation phase was needed as was described in Section 6.3. At the end of the phase, the TSI code was deemed verified, since it showed similar results to the RKF propagation.

7.3.5. PHASE 5: COASTING AND CIRCULARISATION

Once the TSI method was verified, the rest of the trajectory propagation could be added. For the coasting phase this meant that the thrust had to be set equal to zero at the point of main engine cut-off. Fortunately, this was a simple change in class value for the TSI method, however this did not work for RKF. Instead it was discovered that because of the manner in which the state derivative class is set-up and used in the integrator itself, the class had to be re-initialized if anything in the state derivative class changed. Therefore, a new integration had to be started for RKF at main engine cut-off.

Because the circularisation is treated as an instantaneous burn, this only affected the total propellant mass and velocity of the MAV. Therefore, it could be treated with simple circularisation equations described by Wakker (2010). The velocity required to reach a perfect desired circular orbit can be computed. Then because the equations are simple and can be solved by hand, the code could be verified (and validated) by using simple cases computed using the program and then compared to the hand written solutions.

7.3.6. PHASE 6: TOOL VALIDATION

The most important part of this thesis is the comparison between RKF and TSI, therefore these two methods should produce results that can be compared and where the only difference is the manner in which they were computed. Provided that the RKF method is verified and validated,TSI could be validated using RKF with this purpose as mentioned before. However, besides this validation, it should also be checked if these solutions represent a real-life solution to the problem. This means validating both methods against other validated simulations outcomes from other people. Unfortunately, no flight data is available for a Mars Ascent case yet. Two JPL internal sources were found that could potentially be used to validate the software: Woolley (2015) and Benito and Johnson (2016). Woolley used an analytical model to approximate an ascent trajectory for different launch cases and conditions, one of which was a SSTO launcher which was used for validation. Benito on the other hand used a high-fidelity simulation program to simulate the ascent trajectory of some of the newer concept MAV designs numerically. Some of the latest data was provided and was also used for the validation.

In Table 7.2 are the validation numbers provided associated with the simulated trajectory of the case as presented by Woolley. The left columns show the inputs that were provided and the inputs that were used in the program. The right columns show the results of the simulation as provided and as computed by the program. The thrust angles in that column represent the required thrust angles that were needed as input to get the desired outcome.

Table 7.2: Validation case 1: SSTO Woolley

Provided parameters	**Given**	**Used**	**Provided parameters**	**Given**	**Computed**
Thrust [kN]	3.56	3.56	Propellant mass [kg]	202.2	202.07
I_{sp} [s]	256	256	Burn-out angle [deg]	6	5.344
Burn Time [s]	142.6	142.5	Circularisation ΔV [m/s]	181	159.6
MAV Gross Lift-Off Mass (GLOM) [kg]	267.4	267.4	Ascent time [s]	1757	1796.27
Launch altitude [km MOLA]	-	-0.6	Thrust azimuth [deg]	-	-0.299
Launch latitude [deg]	0.0	0.0	Thrust elevation [deg]	-	-0.184
Launch longitude [deg]	-	74.5			
Launch ground velocity [km/s]	-	$1 \cdot 10^{-5}$			
Launch Flight-path angle (FPA) [deg]	90	89			
Launch Azimuth [deg]	90	90			
Desired altitude [km MOLA]	390	390			
Desired inclination [deg]	45	45			

It can be seen that there are some differences, however in this case the validation is to prove that the trajectory computed by the simulation program is indeed a viable trajectory (so close to a reference trajectory). This is because that is enough to compare the RKF and TSI methods. The plotted trajectories are shown in Figures 7.1 to 7.4.

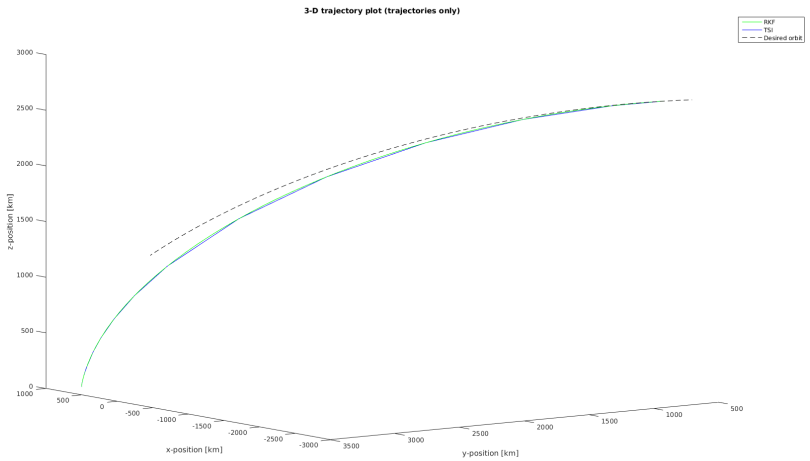


Figure 7.1: Case 1 3-D trajectory

A similar validation was done using the second case. However, in this case the reference trajectory was provided. But the main difference here was that the simulation program still was only able to use a constant thrust elevation and azimuth angle. Therefore, the exact profile could not be matched. Benito also included extra perturbing effects creating another difference. However, it was possible to get a trajectory close to the one provided, which is enough for all current purposes. Table 7.3 shows the used values and computed parameter values for case 2 (similar to case 1).

Again, the trajectory was visualised, however, this time the reference trajectory was also known. In the figures this is represented by a red line. These are shown in Figures 7.5 to 7.8.

The difference in trajectory can mainly be explained by the thrust elevation and thrust azimuth angle

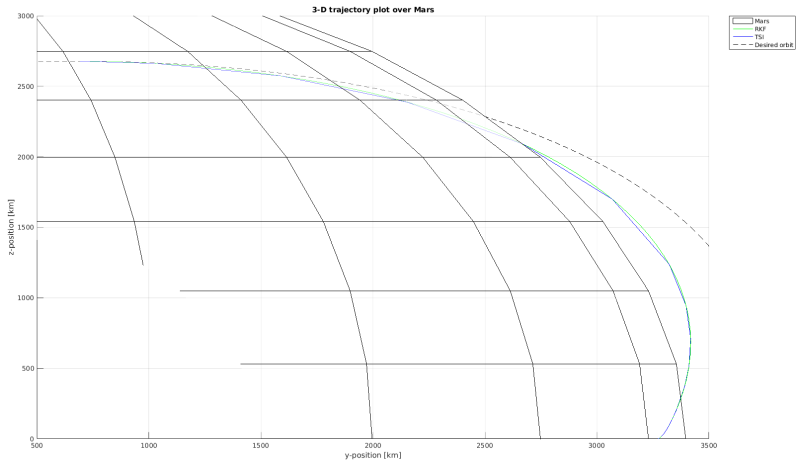


Figure 7.2: Case 1 2-D trajectory as seen from the x-axis

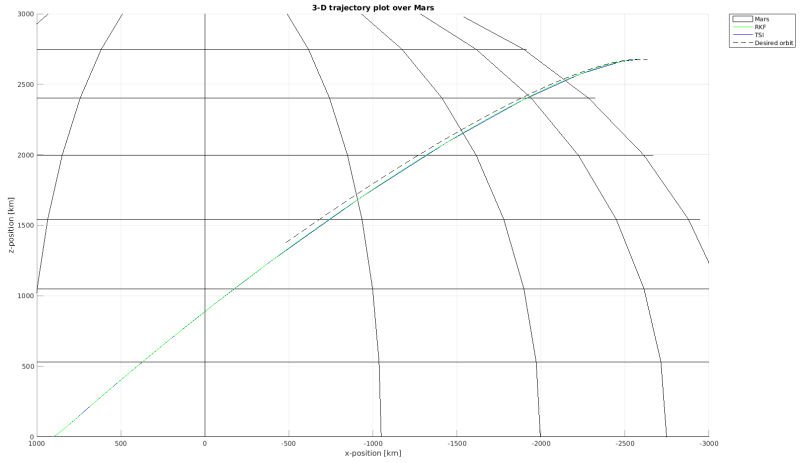


Figure 7.3: Case 1 2-D trajectory as seen from the y-axis

Table 7.3: Validation case 2: SSTO Benito

Provided parameters	**Given**	**Used**	**Provided parameters**	**Given**	**Computed**
Thrust [kN]	6.1087	6.0187	Propellant mass [kg]	213.85	213.85
I_{sp} [s]	315.9	315.9	Burn-out angle [deg]	-	15.644
Burn Time [s]	99.362	99.362	Circularisation ΔV [m/s]	757.57	594.74
MAV GLOM [kg]	288.96	288.96	Ascent time [s]	949.12	876.09
Launch altitude [km MOLA]	-0.8	-0.6	Thrust azimuth [deg]	-	-0.7273
Launch latitude [deg]	0.0	0.0	Thrust elevation [deg]	-	-0.674
Launch longitude [deg]	90	90			
Launch ground velocity [km/s]	$1 \cdot 10^{-6}$	$1 \cdot 10^{-6}$			
Launch FPA [deg]	90	89			
Launch Azimuth [deg]	90	90			
Desired altitude [km MOLA]	479.19	479.19			
Desired inclination [deg]	92.69	92.69			

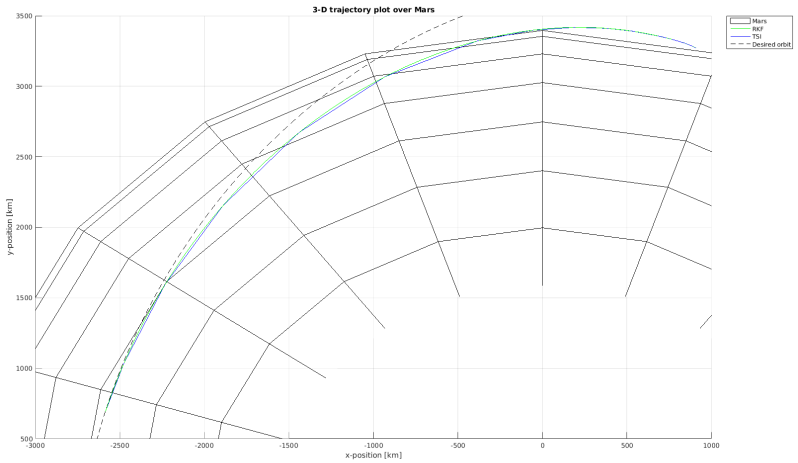


Figure 7.4: Case 1 2-D trajectory as seen from the z-axis

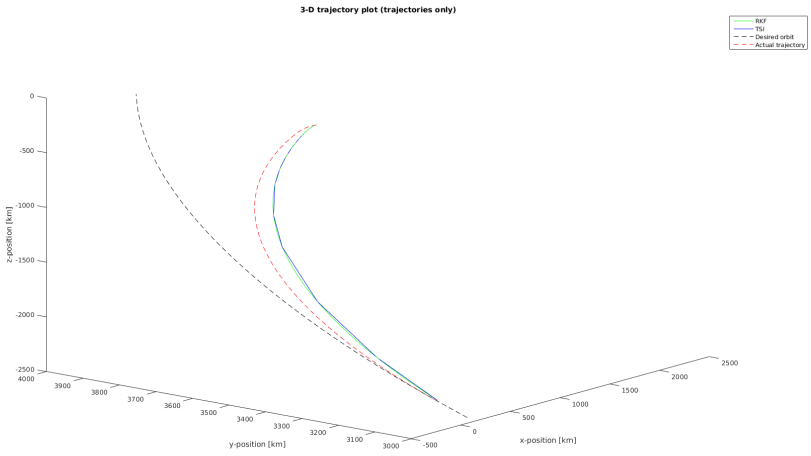


Figure 7.5: Case 2 3-D trajectory

curve of the reference trajectory. This curve was derived from the thrust data provided by Benito. This curve is shown in Figure 7.9.

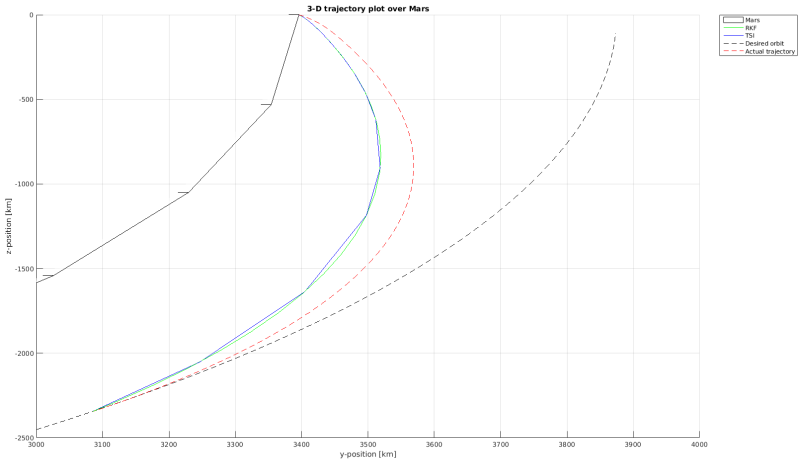


Figure 7.6: Case 2 2-D trajectory as seen from the x-axis

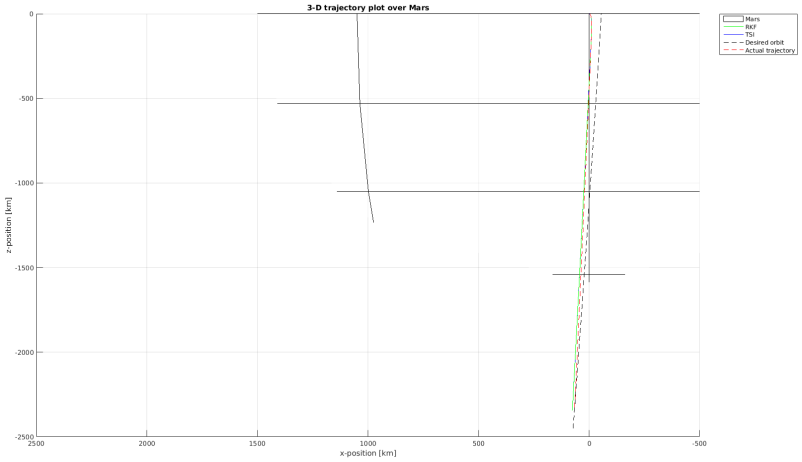


Figure 7.7: Case 2 2-D trajectory as seen from the y-axis

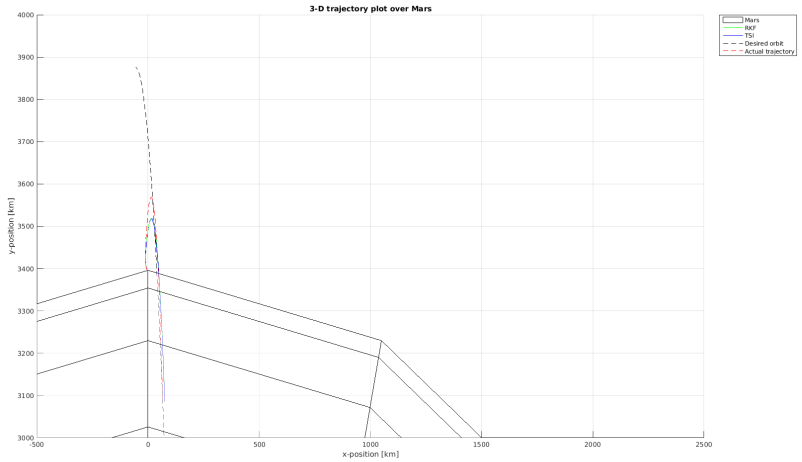


Figure 7.8: Case 2 2-D trajectory as seen from the z-axis

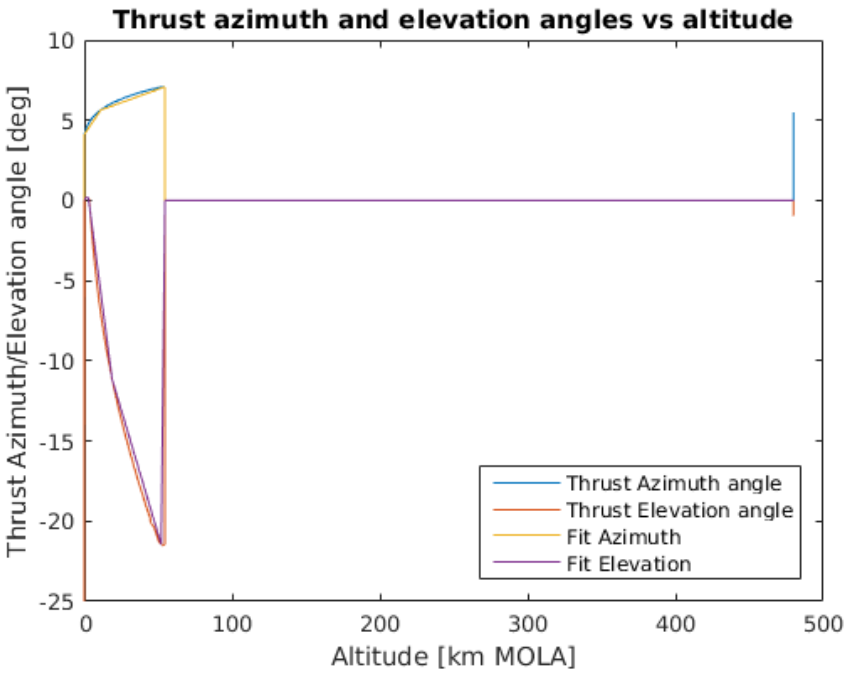


Figure 7.9: Case 2 thrust elevation and azimuth angle curve.

8

RESULTS AND ANALYSIS

With the program validated, both case 1 and case 2 can be used to examine the behaviour of TSI. This is done in a sensitivity analysis and is very useful to determine the limitations and characteristics of this method. In the previous chapter the trajectories from both TSI and RKF were plotted and it could be seen that the results were very similar to the point where the accuracy of TSI was validated. Therefore, in this chapter the performance of TSI will be compared to RKF to determine any advantages and disadvantages. In order to do this only one variable will be changed every time, which means that from the nominal case only one parameter will be different. This way the effects can be clearly associated with that particular parameter, which provides a solid base of comparison. The nominal values for both the first and second case are provided in Appendix E. It is also important to have the same cut-off point. This can either be a certain altitude, set end time or zero degrees flight-path angle. Usually the set end time is taken as the cut-off criteria to determine the difference in state. Sometimes the altitude is used in cases where it is interesting to see when a certain altitude is reached and under what conditions when a certain parameter is changed. An example was shown in the validation chapter where certain conditions had to be met at a certain altitude. And in some cases it is best to set the zero degrees flight-path angle as a cut-off criteria because either the cut-off time or altitude will never be reached but the effects of a parameter still need to be investigated. This last cut-off criteria is the "fail-safe" for every simulation because this prevents the simulation from going back to the surface and literally crashing the program. In this chapter every section discusses the effect of a different parameter or focusses on one aspect of the simulation.

Besides comparing TSI to RKF it will also be compared to the non-rotational case. The nominal case takes the rotating Mars into account, however it is interesting to see what TSI does when the rotation is turned off. For these two cases to be properly compared, again for most parameters, the set end time is the cut-off criteria unless mentioned otherwise.

The parameters changed in this sensitivity analysis are: order, error tolerance, launch altitude, launch latitude, launch longitude, flight-path angle and heading angle. A multiple run analysis will also be described where the nominal case is run for a number of times and the CPU time is compared.

8.1. ORDER

For most classic integrators the order is set and cannot be easily changed. For RKF for instance, a whole new set of coefficients is required if a different order is desired. However, for TSI changing the order is as simple as updating the maximum order input value. Theoretically this means that an infinite order can be chosen to get the most accurate result. The question is, is this indeed the most optimal. Section 8.1.1 hopes to answer this question by looking at the CPU time. In Section 8.1.2 a range of orders has been run for both cases. Once with the nominal conditions of a rotating Mars and once assuming a non-rotating Mars. The effect of the rotating Mars will be investigated. The cut-off criteria for case 1 was an end time of 1796 seconds for the data collection case and 789 seconds without live data collection. For case 2 the cut-off criteria was 876 second for both runs.

8.1.1. OPTIMAL ORDER

To determine the optimal order an important criteria is that the outcome is accurate, however, what is often more important is that this accurate outcome is computed really fast. This is why, in order to find the optimum order, the CPU time is the most important criteria. In order to properly observe the effect of a different order, the nominal case is run for different orders. Thus the only difference will be the maximum order. This can be done for the two presented cases. During real-life simulations a user can choose to either collect all the data during the simulation or only store the final outcomes at the end of the simulation. Storing data during the integration requires a lot of computational power and will therefore increase the CPU time required, but this will provide the user with the desired trajectory data. Both cases with and without live data collection have been run for a specific range of orders depending on the case, and the results are shown in Figure 8.1. The first case was run with a maximum order ranging from 5 to 100, and the second case was run from 5 to 31 because of the behaviour observed during the first case runs.

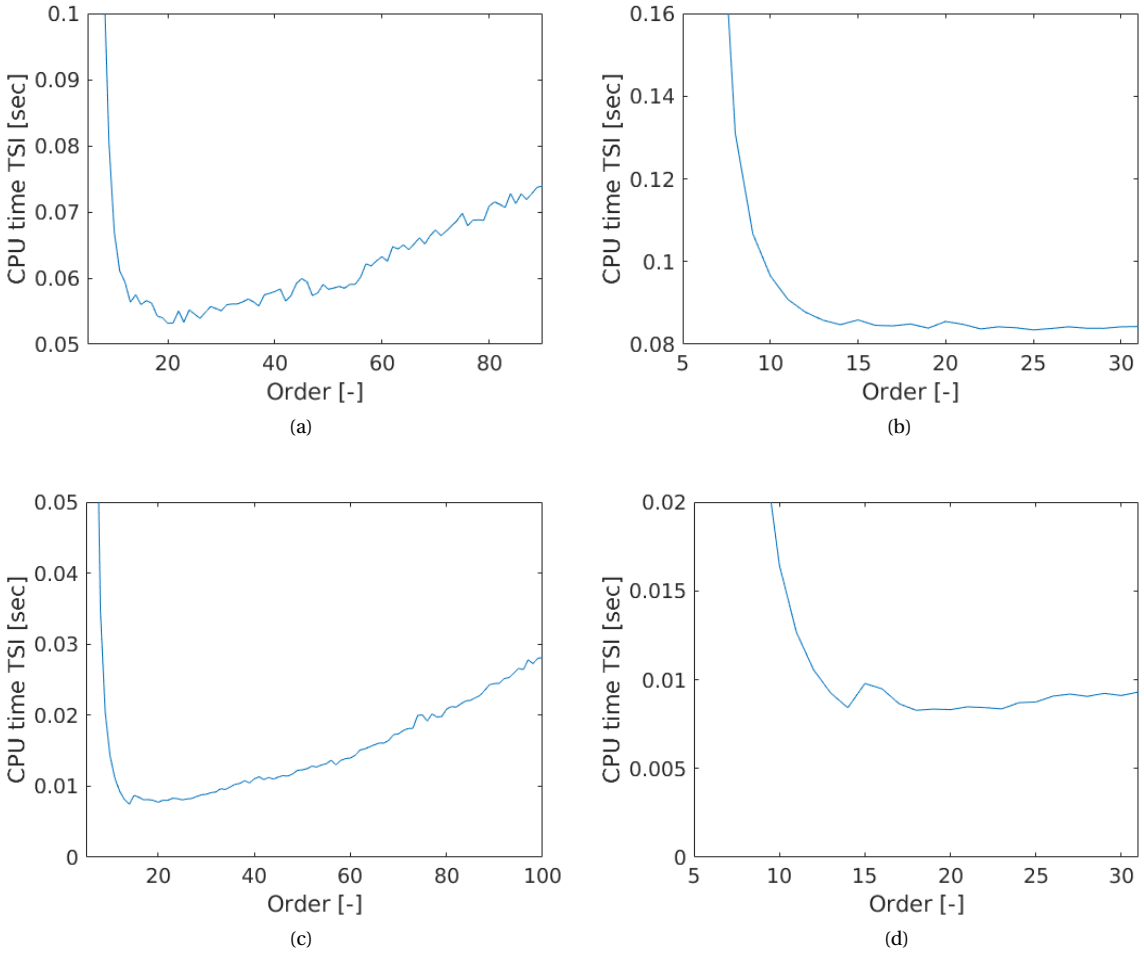


Figure 8.1: Order versus CPU time (a) case 1 including live data collection, (b) case 2 including live data collection, (c) case 1 without live data collection, (d) case 2 without live data collection

Looking at Figure 8.1a it is clear that the order versus CPU curve can be split up into two phases that have different properties that influence the CPU time. The first phase shows a rapid decrease in the CPU time until order 20 is reached, the second phase continues from that point until the final order is reached. The order has a direct relation to the chosen step-size that TSI computes automatically because the step-size is determined by the error tolerance and the difference between the penultimate Taylor Series coefficient and the final one (as described in Section 5.1.2). This means that if these last two coefficients show a large difference, which can happen lower order because fewer derivatives are taken into account, the step-size will be lowered and thus increasing the number of steps. In the first phase, this large number of steps result in a high CPU time.

Another factor that affects the CPU time is the number of derivatives taken into account, which is directly related to the order (10 as maximum order equals 10 derivatives that are taken into account). At a certain point, an increase in order does not affect the number of steps that much any more, resulting in a decrease of 3 to 1 steps with every order increase. The result is that the number of steps remains approximately the same but the number of derivatives that are taken into account keeps increasing. Every extra derivative means an extra computation per auxiliary equation/variable. This is what is observed in the second phase. Here the effect of adding more computations on the CPU time becomes clear, showing a corresponding increase in CPU time. This tipping point between the two phases can be observed around the lowest CPU time, where the effect of the number of steps and the number of derivative computations equal out. This is therefore the optimum order for the TSI method. The two phases can be observed in each of the four graphs, however, because the increase of CPU time, the higher orders were not required and were thus left out in the simulations for case 2. For each of the simulation runs, the two orders which resulted in the lowest CPU time have been recorded in Table 8.1.

Table 8.1: Lowest two orders with corresponding CPU times for each of the sub-graphs

Sub-graph	Lowest CPU time		Second lowest CPU time	
	Order	CPU time [s]	Order	CPU time [s]
a	20	0.053162	21	0.053226
b	25	0.083474	22	0.0837
c	14	0.007465	20	0.007736
d	18	0.008273	20	0.008314

In the table and the graphs it can be seen that the minima are all situated around an order of 20, which is why a maximum order of 20 was set as the nominal value and used for all simulations.

8.1.2. COMPARISON WITH NON-ROTATING MARS

For the comparison of the rotating and the non-rotating cases with a varying order it is again interesting to look at the differences in CPU time. Figure 8.6 shows these plots for both cases when the trajectory data is not collected during the simulation. It can be seen that for case 1, on average, the non-rotating runs are faster. Also, for a non-rotating Mars the optimum order is 20 as well. The difference in CPU time is better shown by the case 2 run, where the non-rotating curve fits perfectly below the rotating curve. Here, the order 20 has again the second lowest CPU time (the lowest is order 17 with a time difference of 28 microseconds). The time difference between the curves of the rotating and non-rotating Mars can be explained by the fact that if Mars does not rotate, the rotating effects become zero, which reduces the number of computations.

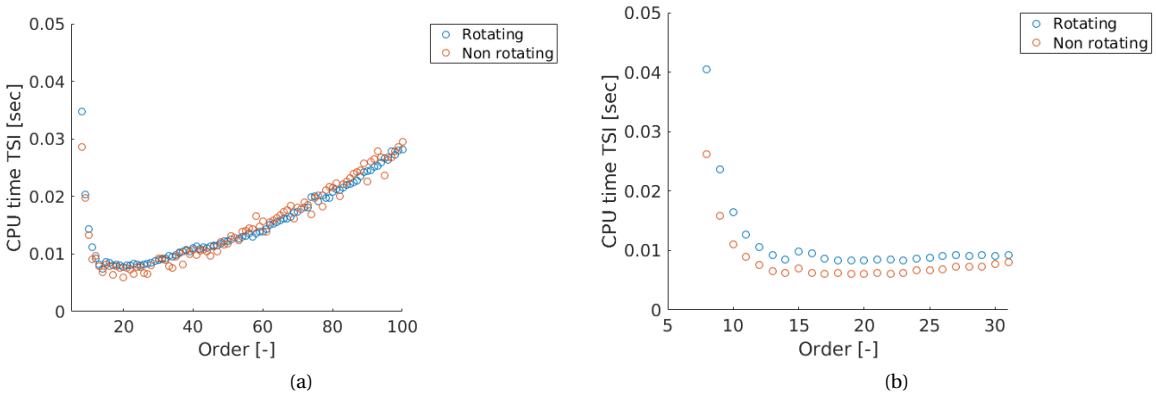
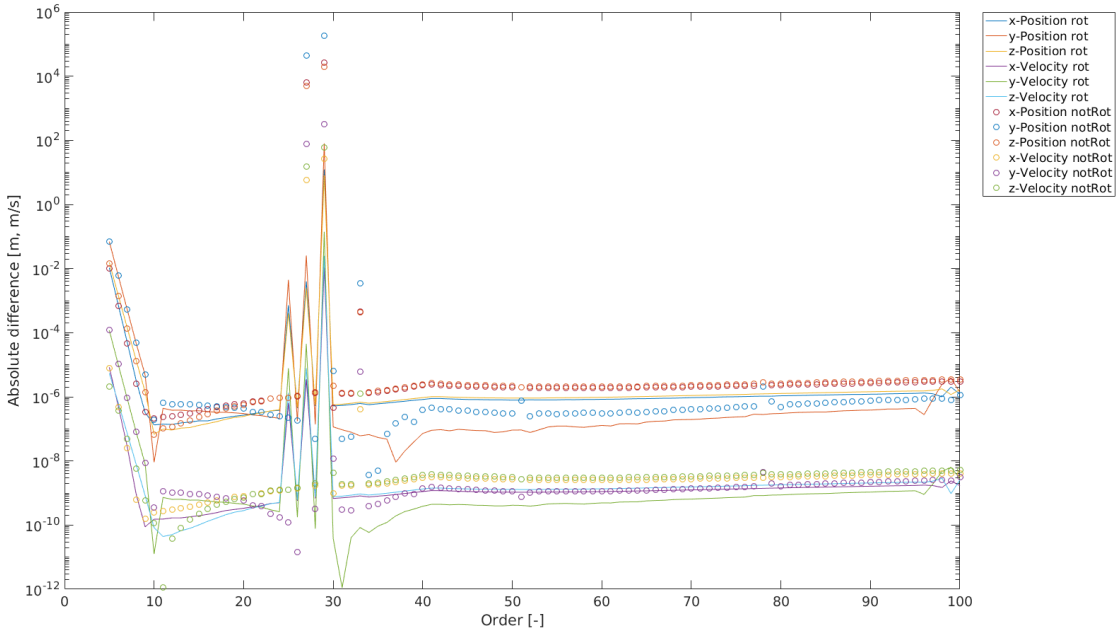


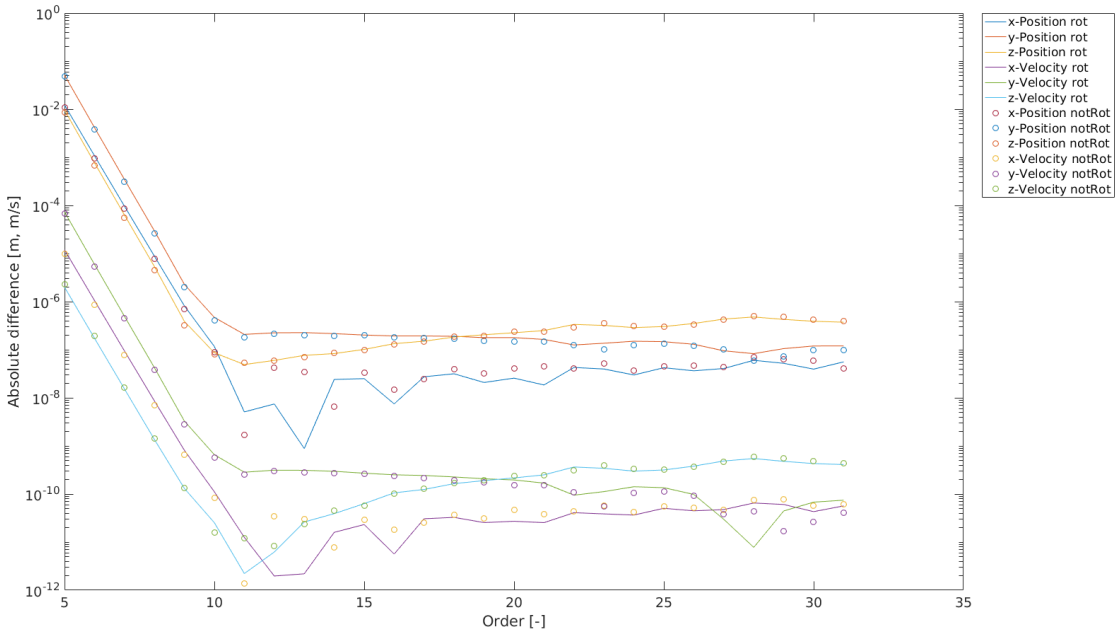
Figure 8.2: Order versus CPU time for rotating and non-rotating Mars (a) case 1, (b) case 2

As was mentioned before, the accuracy of the results is very important. In Figure 8.3 the absolute difference between the nominal end state of RKF before circularisation and TSI for both the rotating and non-rotating Mars. A trend can be identified where it is shown that an order of 11 or higher will result in a difference

of μm in position and nm/s in velocity. Even the lowest order of 5 results in an accuracy of less than 10 cm for position and 10 mm/s for velocity. The spikes in the first case are caused by a slight error resulting from the combination of variables that result at those specific orders. This proves that the simulator is not yet robust enough and could still use some improvements. One improvement that might solve this problem is a change in the manner in which the step-size is computed. Additional comparison plots are shown in Appendix F.



(a)



(b)

Figure 8.3: Difference with respect to RKF case for rotating and non-rotating Mars (a) case 1, (b) case 2

8.2. ERROR TOLERANCE

The error tolerance is used to determine the required step-size during the integration. A high error tolerance (say 10^{-5}) will therefore provide an end result that is less accurate than a lower error tolerance (such as 10^{-15}). For the RKF78 provided by Tudat the lowest accuracy that can be used is 10^{-15}, which is why that accuracy is used as the nominal accuracy. When a range of error tolerances is examined the accuracy of the end state and the speed of convergence can be determined. The speed of convergence follows from the consecutive differences between the end states and the accuracy of the end states comes from the comparison to the nominal case. Both these characteristics will be examined and compared for RKF and TSI, see Section 8.2.1, and for the rotating and non-rotating Mars described in Section 8.2.2. The cut-off criteria for case 1 was 789 seconds and 876 seconds for case 2 and both cases were run without live data collection.

8.2.1. COMPARISON WITH RKF78

The speed of convergence is important because it shows the relationship between the different error tolerances. If the differences are small it means that the change in error tolerance had a small effect. If the consecutive difference is large, it means that the solution has not yet converged. Figure 8.4 shows the consecutive differences in position and velocity for the two cases for both integration methods. Here it can clearly be seen that TSI converges much faster than RKF. The curve of RKF lies quite a bit higher, which means that the differences are larger, and thus it converges slower than TSI.

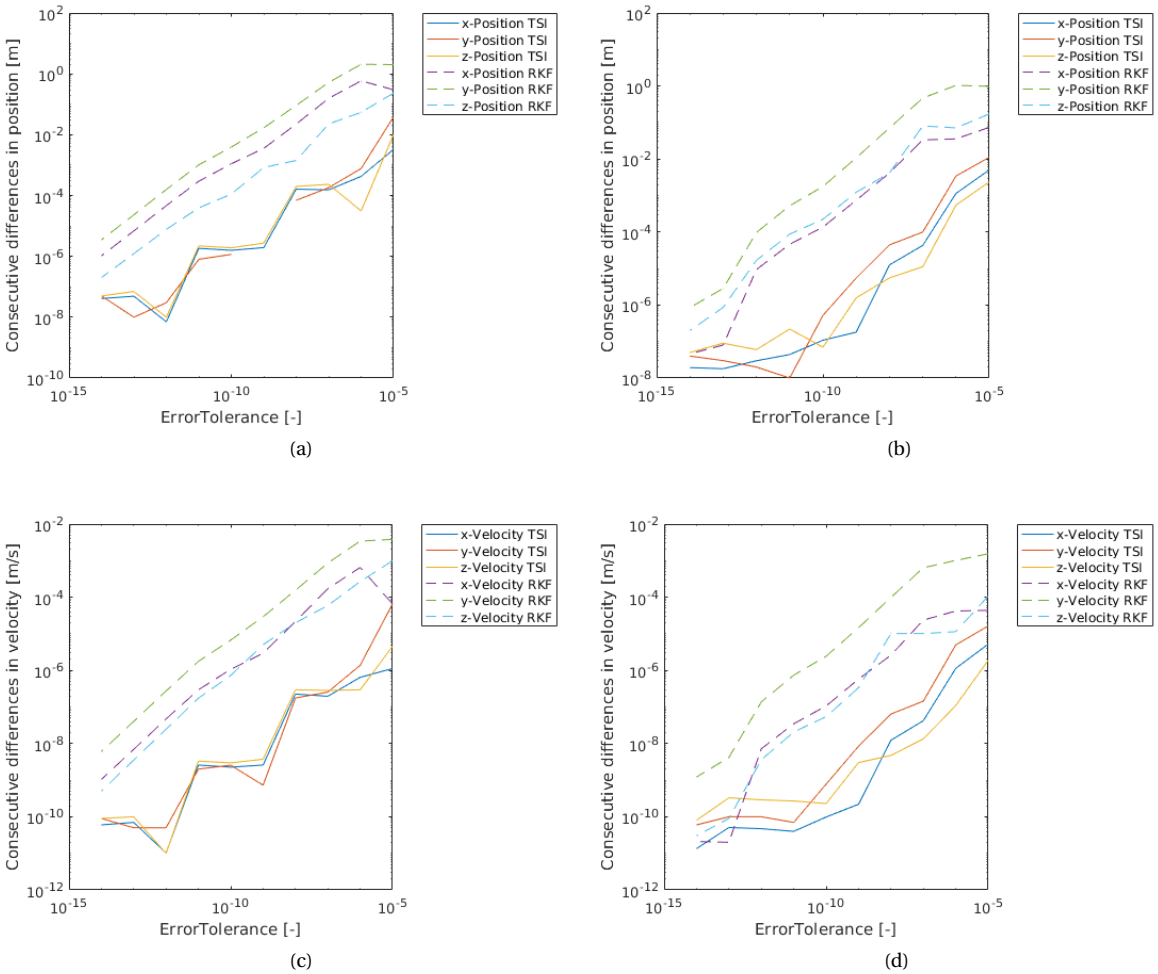


Figure 8.4: Error tolerance versus consecutive difference between end states before circularisation for TSI and RKF (a) case 1 position, (b) case 2 position, (c) case 1 velocity, (d) case 2 velocity

A similar comparison can be done when comparing each of the end states directly to the nominal end

state. The result is a nominal absolute difference graph as presented in Figure 8.5 for position and velocity and both cases. Here it can be seen that the accuracy of the position at an error tolerance of 10^{-5} is already in cm for TSI whereas this accuracy is only reached at an error tolerance of 10^{-9} for RKF. For the velocity the TSI accuracy is in the order of 10 μm at this initial error tolerance which is reached at an error tolerance of 10^{-9} using RKF. This means that TSI is, on average, more accurate at any given error tolerance.

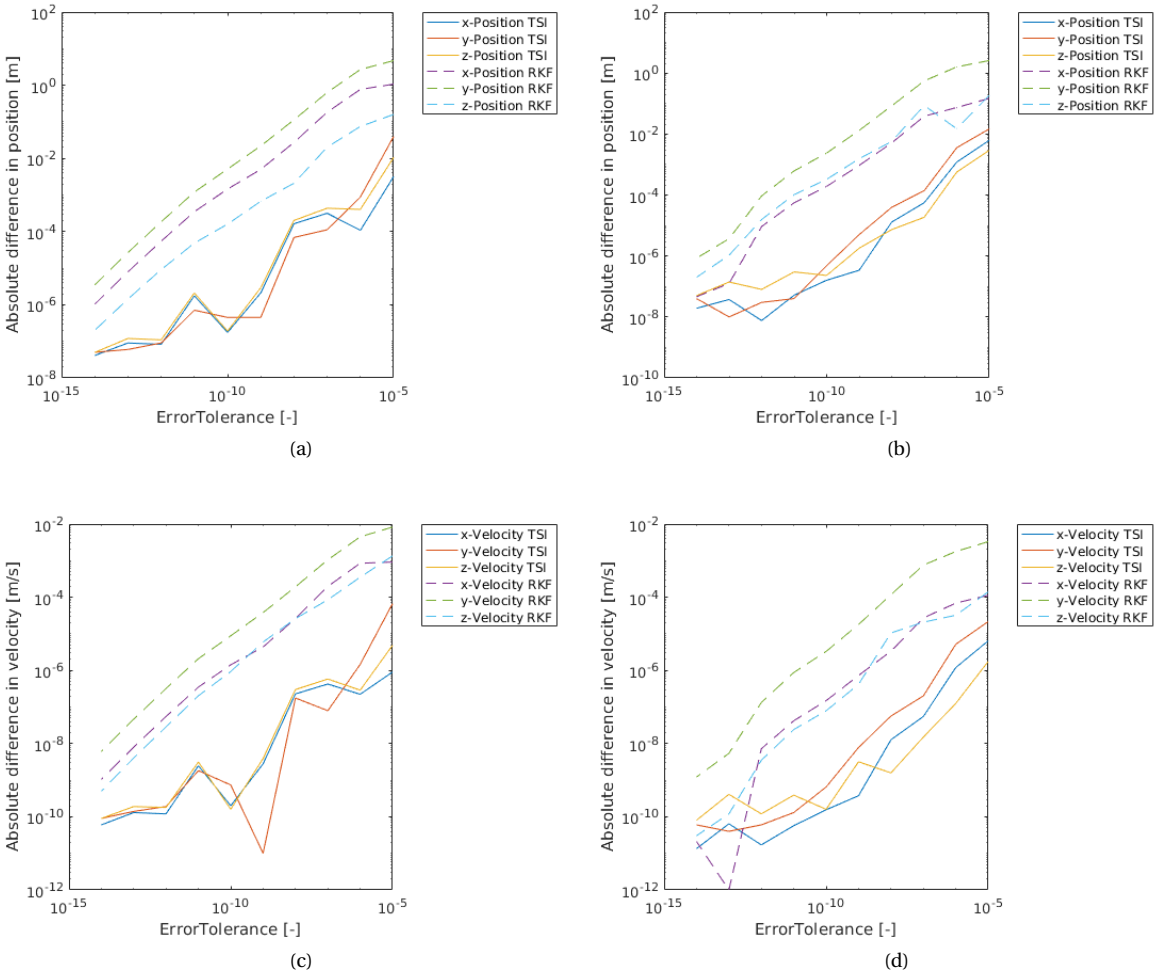


Figure 8.5: Error tolerance versus nominal absolute difference between end states before circularisation for TSI and RKF (a) case 1 position, (b) case 2 position, (c) case 1 velocity, (d) case 2 velocity

Another important observation can be made when looking at the difference in function evaluations and CPU time for each error tolerance for both RKF and TSI. The values for both cases are shown in Table 8.2.

Table 8.2: Number of function evaluations and CPU time for RKF and TSI

Error tolerance	**Case 1**				**Case 2**			
	Number of evaluations		CPU time [ms]		Number of evaluations		CPU time [ms]	
	RKF	**TSI**	**RKF**	**TSI**	**RKF**	**TSI**	**RKF**	**TSI**
10^{-5}	169	24	0.419	4.703	156	24	0.392	5.353
10^{-6}	182	24	0.432	4.57	182	25	0.49	5.518
10^{-7}	221	26	0.542	4.769	208	27	0.527	5.823
10^{-8}	260	29	0.614	5.272	247	29	0.567	6.354
10^{-9}	312	30	0.823	5.571	286	30	0.709	6.439
10^{-10}	364	32	0.845	5.685	338	33	0.789	7.083
10^{-11}	455	36	1.032	6.227	390	36	0.904	7.454
10^{-12}	546	38	1.277	6.506	546	39	1.231	8.02
10^{-13}	702	41	1.541	7.062	728	41	1.604	8.627
10^{-14}	910	42	1.965	7.227	936	45	2.055	9.538
10^{-15}	1222	46	2.666	7.754	1300	47	2.851	9.538

Here one evaluation is defined as running through the derivative equations once. For TSI this means that each time step, one evaluation is performed because all desired derivatives are computed during one computational run. In the case of RKF78 however, each time step consists of 13 computational runs. Table 8.2 shows that for both cases the number of evaluations of RKF are approximately one order more at the high error tolerances but almost two orders higher at the lowest tolerance compared to TSI. This means that a decrease in error tolerance results in a large increase in number of evaluations for RKF but a relatively small increase for TSI. For both cases, the number of evaluations is not even doubled from the highest to the lowest tolerance, whereas the maximum number of evaluations for RKF is 7 to 8 times more.

This advantage however does not translate well to the CPU time where it can be seen that for most of the error tolerances RKF is an order faster than TSI. The similar increase in CPU time for both integrators is nevertheless still observed. An explanation for the large difference in CPU time could be that TSI was not optimally programmed and thus results in a relatively high CPU time. Further analysis is required.

8.2.2. COMPARISON WITH NON-ROTATING MARS

Looking at the rotating Mars and non-rotating Mars runs, a noticeable difference in the CPU time can be observed as shown in Figure 8.6. In both cases the non-rotating Mars run is approximately 0.2 ms faster. This is similar to what was observed with the order runs.

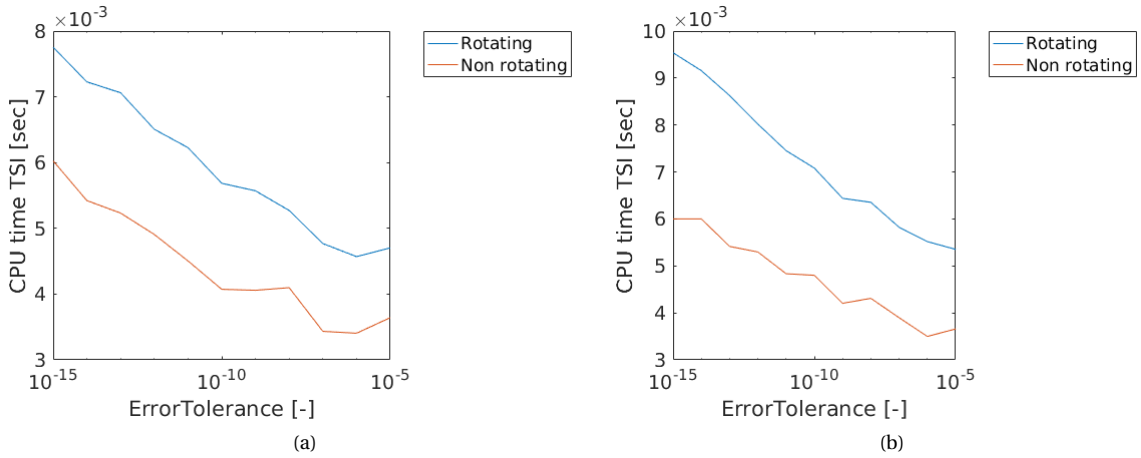
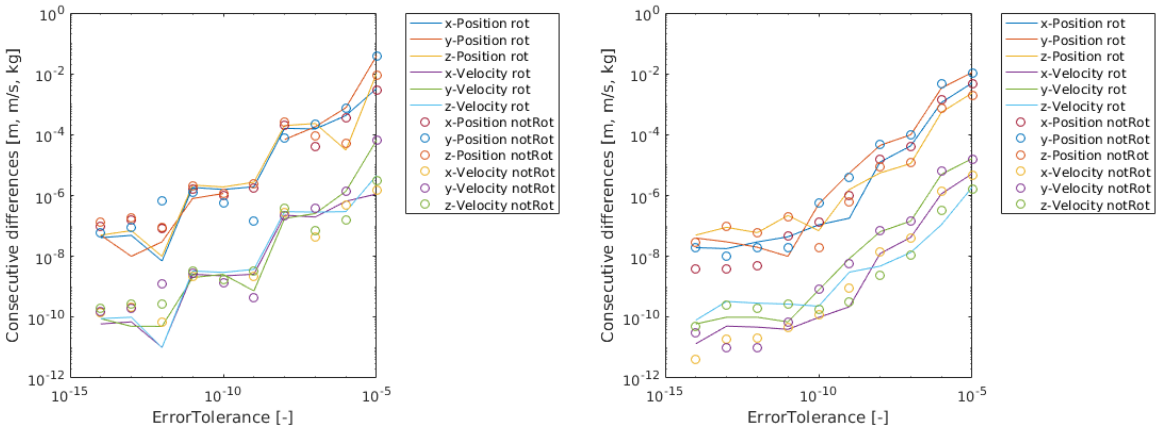


Figure 8.6: Error tolerance versus CPU time for rotating and non-rotating Mars (a) case 1, (b) case 2

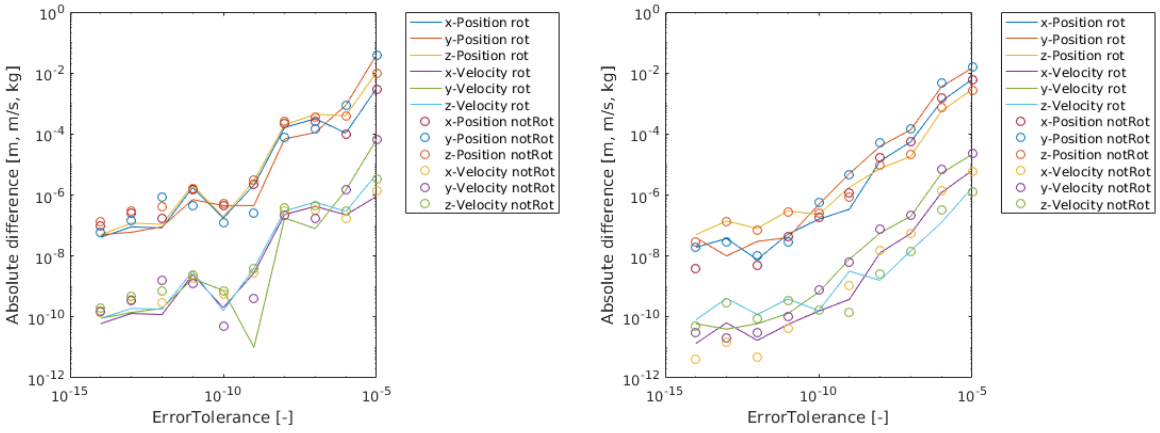
The real question is, is there a noticeable difference in the convergence speed and the accuracy? This

question can be answered by looking at Figures 8.7 and 8.8. Here it can be seen that the rotation of Mars does not have a significant influence on the convergence speed nor the accuracy of the results.



(a) (b)

Figure 8.7: Error tolerance versus consecutive difference for rotating and non-rotating Mars (a) case 1, (b) case 2



(a) (b)

Figure 8.8: Error tolerance versus nominal absolute difference for rotating and non-rotating Mars (a) case 1, (b) case 2

8.3. MULTIPLE RUNS

In Section 8.2.2 it was shown that in this thesis research, the TSI turned out to be slower than RKF for both tested cases. This in contrary to the reference research. Therefore it is a good idea to look at the differences that could occur when measuring CPU time for different runs. In this section the nominal case has been run 5000 times in sequence for both RKF and TSI (Section 8.3.1) using a rotating and a non-rotating Mars (Section 8.3.2). If the computer and the CPU measurements would be perfect, there should be no difference in CPU time and there would simply be two lines. This, interestingly enough, is not the case.

The graphs shown in this section can also be found enlarged in Appendix F.

8.3.1. COMPARISON WITH RKF78

Again the notion that RKF is faster than TSI is confirmed when looking at Figure 8.9, however for both cases there is not one single line per integrator. Instead several outliers and patterns can be observed.

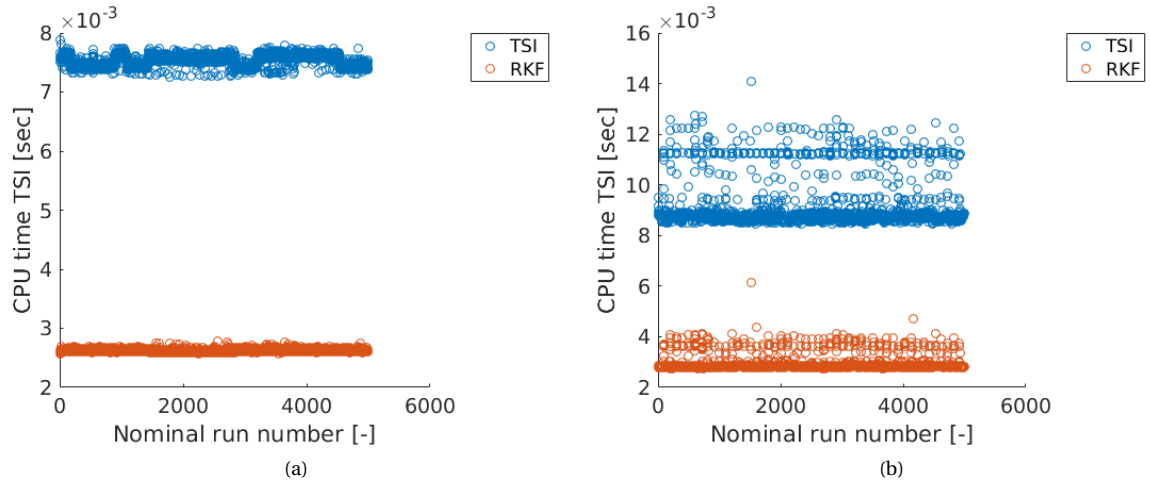


Figure 8.9: Nominal runs versus CPU time comparison between RKF and TSI (a) case 1, (b) case 2

For case 1 it can be seen that there were some periods where the CPU time was slightly higher. This is even clearer for case 2 where two different CPU levels can be distinguished and many random CPU times in between. The CPU time is computed using the internal clock of the computer. And it could occur that background programs are run while the integration is running. In such a case, the CPU time computed for that particular integration run can turn out to be higher than expected because the internal clock cannot differentiate between the simulation program and all other programs running at that same instance. Therefore, even if the same nominal case is run, the CPU time could still be 30% higher or more.

8.3.2. COMPARISON WITH NON-ROTATING MARS

The effect of a program running in the background during a simulation run becomes even more clear when looking at the rotating and non-rotating Mars CPU times shown in Figure 8.10.

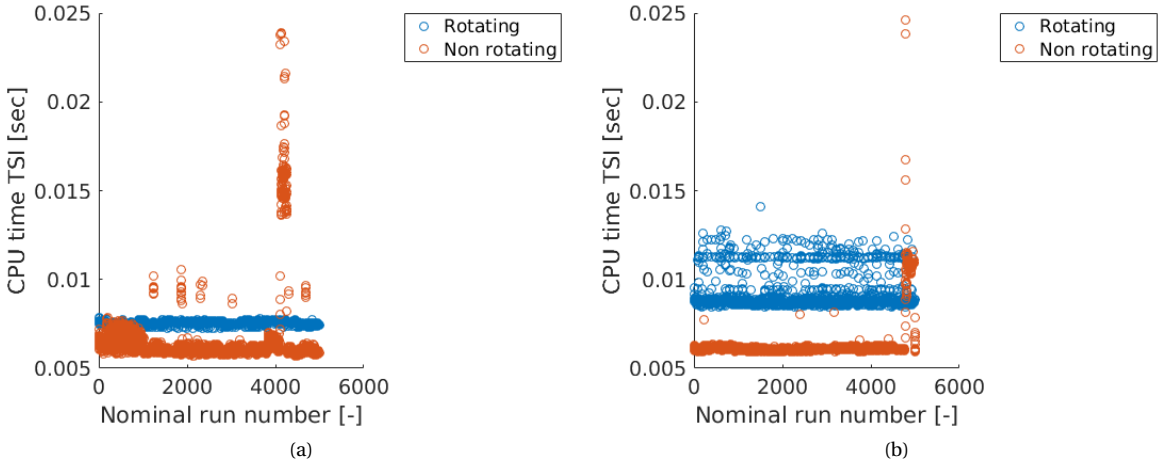


Figure 8.10: Nominal runs versus CPU time for rotating and non-rotating Mars (a) case 1, (b) case 2

Here it can be seen that, on average, the non-rotating Mars runs are fasters than the rotating Mars runs, which is similar to what has been observed before. However, in both cases there was a spike where the CPU time suddenly increased tremendously for the non-rotating runs. This shows, that at that time a program (or process) was running in the background and clearly affected the performance of the simulation a lot. This means that CPU time is not always an accurate representation of the performance of the simulation. As a matter of fact, if the same program would be run on a different computer, the CPU times could be completely

different. But it would still show the same relationship between the rotating and non-rotating Mars runs as well as RKF and TSI and does therefore not explain why TSI is so much slower compared to RKF.

8.4. LAUNCH ALTITUDE

From now on only the rotating runs and the non-rotating runs will be compared. This can be done because TSI has proven to be accurate compared to RKF producing similar results. Thus it is only interesting to look at the effect of the different parameters on the performance of TSI. In this case the effect on CPU time, consecutive difference and absolute difference will be investigated as well as effect on end radius and end velocity before cirularisation. Here the cut-off criteria for case 1 was an end time of 789 seconds and 856 seconds for case 2.

In Figure 8.11 the launch altitude is plotted against the CPU time. This has been done for the altitude range from the candidate altitude of -0.6 km MOLA to the maximum altitude where the MAV could be launched from; 0.5 km MOLA. In this case, no clear pattern can be observed due to the large differences in CPU time. Therefore it can be concluded that the launch altitude in this particular range does not have an effect on the CPU time.

- Consecutive differences end state before circularisation comb case 1
- Consecutive differences end state before circularisation comb case 2
- Differences nominal case end state before circularisation comb case 1
- Differences nominal case end state before circularisation comb case 2

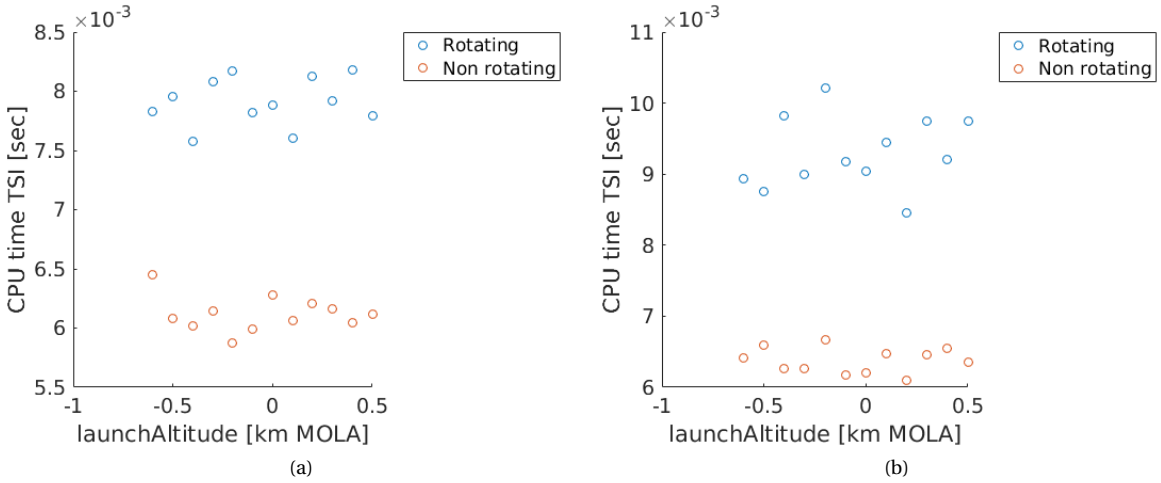


Figure 8.11: Launch altitude versus CPU time for rotating and non-rotating Mars (a) case 1, (b) case 2

Randomness can also be observed when looking at the consecutive difference for the rotating runs in Figure 8.12. Here the z-position and z-velocity show some slight deviations for case 1, and in the curves of case 2 the rotating states all show deviations. However, it is interesting to notice that no deviations are observed for the non-rotating runs, which means that all the deviations are caused by rotating effects. A similar behaviour can be observed in the nominal absolute difference graphs presented in Figure 8.13. Here case 1 shows almost no deviations except for the z-position and z-velocity. However, again large deviations can be observed for case 2. These deviations can be caused by the fact that for case 2 the MAV is launched due East but quickly heads South therefore going against the rotation of Mars and creating variations that are not observed for the non-rotating Mars.

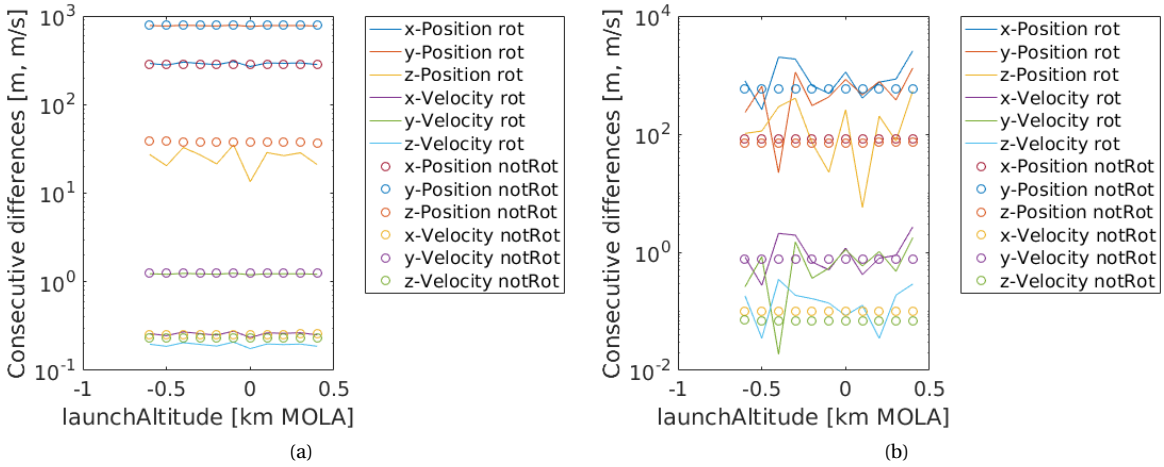


Figure 8.12: Launch altitude versus consecutive difference for rotating and non-rotating Mars (a) case 1, (b) case 2

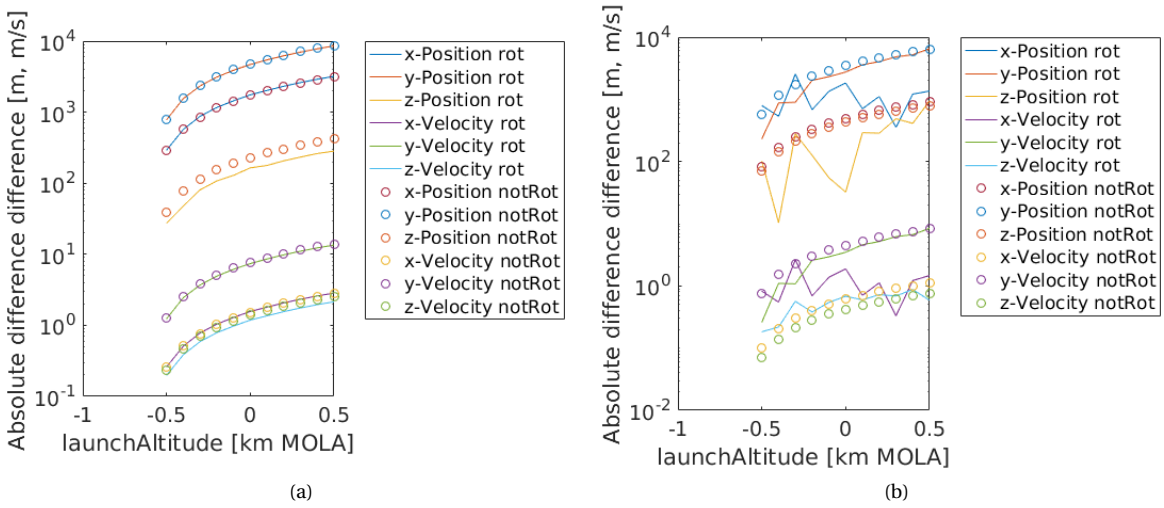


Figure 8.13: Launch altitude versus nominal absolute difference for rotating and non-rotating Mars (a) case 1, (b) case 2

Another example of the effect of the rotating Mars can be seen in the radius graphs in Figure 8.14. Because in both case 1 and 2 the MAV is launched due East on the equator it is able to get an initial ΔV boost. This increase in velocity compared to the non-rotating Mars results in a final radius that is higher than the non-rotating Mars runs for both case 1 and case 2. It can also be observed that the radius of the final orbit increases with an increase in launch altitude. This could be a direct effect of both the drag and the gravitational acceleration, because the atmosphere becomes less dense and the gravitational acceleration becomes smaller the further away from the surface the MAV is.

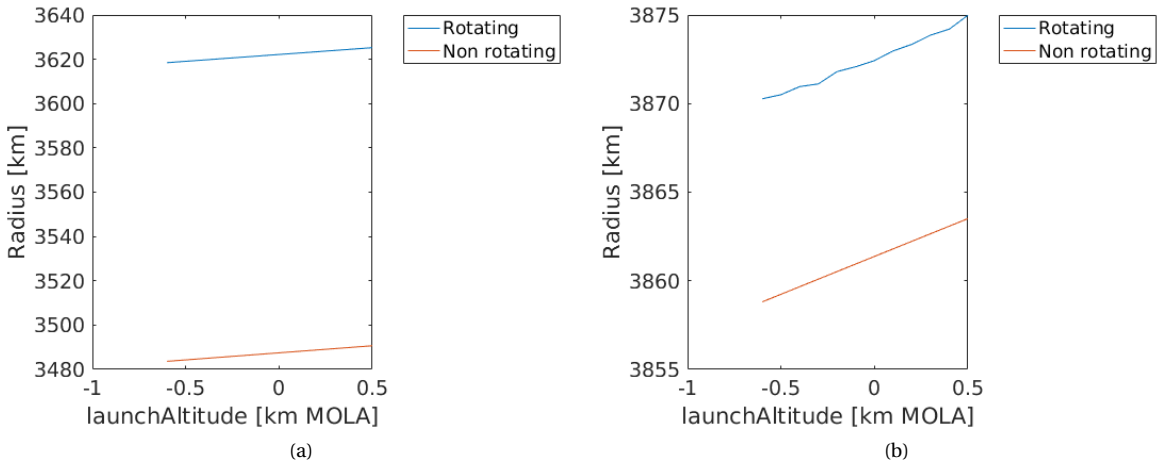


Figure 8.14: Launch altitude versus orbital radius for rotating and non-rotating Mars (a) case 1, (b) case 2

In Figure 8.15 the ground and inertial velocity for both cases and both rotating and non-rotating Mars have been plotted. These are the end velocities of the MAV before circularisation. And because an increase in final radius was observed, a decrease in end velocity was expected and this is indeed observed for both cases.

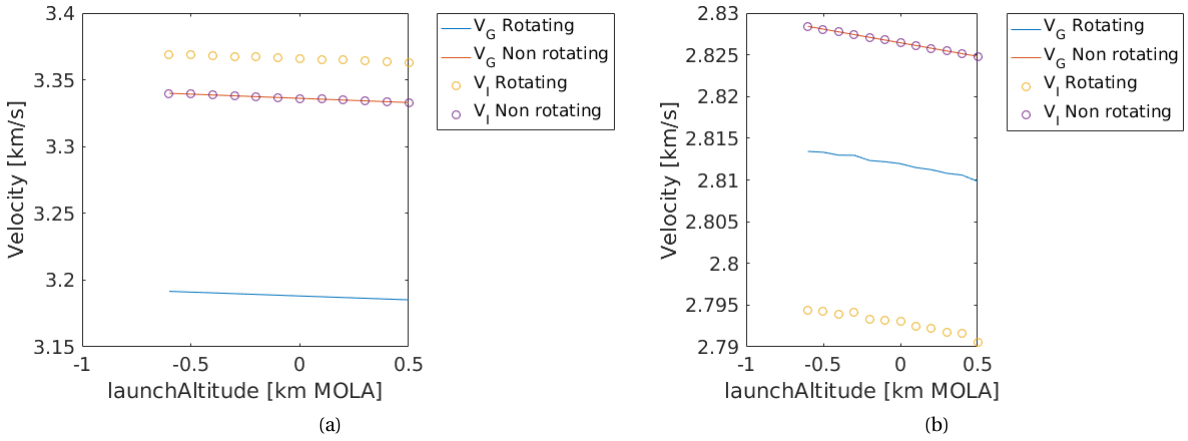


Figure 8.15: Launch altitude versus velocity for rotating and non-rotating Mars (a) case 1, (b) case 2

When comparing ground velocity it would be expected that the rotating Mars run would show a lower velocity because Mars is actually rotating with the MAV. This effectively lowers the speed at which the MAV travels as observed from the Martian surface. This is indeed seen in both cases. What is also noticeable is that there is fortunately no velocity difference between the inertial and ground velocity for the non-rotating case, which means that the rotation of Mars was indeed zero and the program is working properly. There is however a large difference between the ground and inertial velocity for the rotating runs, which is precisely the effective velocity boost from the rotating Mars.

Finally it is interesting to notice the difference in end velocities between case 1 and case 2. Case 1 has an inertial velocity which is higher than the ground velocity and also the velocity for the non-rotating run. This is what one would expect for a MAV that is launched due East. However, case 2 also launches due East and shows an inertial velocity for the rotating run which is lower than all other velocities. This can be explained by taking to account that these are all final velocities before the circularisation and remembering the trajectory profile of both cases as presented in Chapter 7. The trajectory of case 1 takes the MAV slightly North after launch, whereas in case 2 the trajectory goes South and even slightly West. This means that it is moving against the rotation of Mars and thus effectively loses some velocity when compared to both the ground

velocity as well as the velocity in the non-rotating run. This effect was therefore expected. More position and velocity graphs can be found in Appendix F.

8.5. LAUNCH LATITUDE

For the latitude comparisons, the CPU time, radius and velocity are compared. Additional graphs for the end states are provided in Appendix F. The cut-off criteria was an end time of 789 seconds for the first case and 783 seconds for the second case. Again starting with the CPU time as shown in Figure 8.16. Even though the curves from case 1 and case 2 might look different, they show the same behaviour. Starting at the equator and ending on the North-pole, both graphs show that the CPU time of the rotating and the non-rotating case converge until they are practically the same on the North-pole. This clearly shows the effect of the rotating Mars on both cases, because this effect becomes smaller with an increase in latitude.

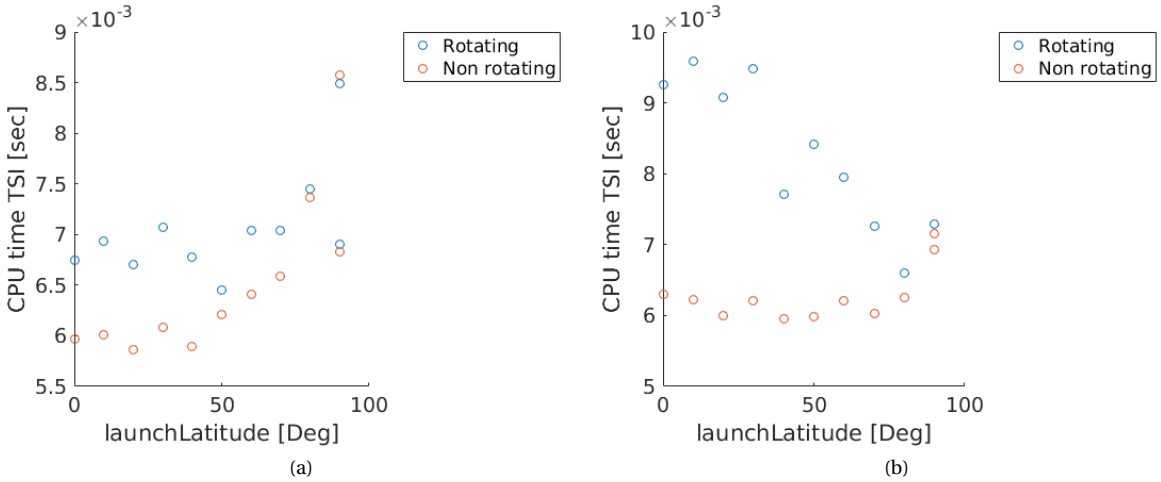


Figure 8.16: Launch latitude versus CPU time for rotating and non-rotating Mars (a) case 1, (b) case 2

Figure 8.17 shows the radius for each of the launch latitudes for both cases. Again, as expected the launch latitude does not have an effect on the non-rotating runs, however it does seem to have a large effect on the rotating runs.

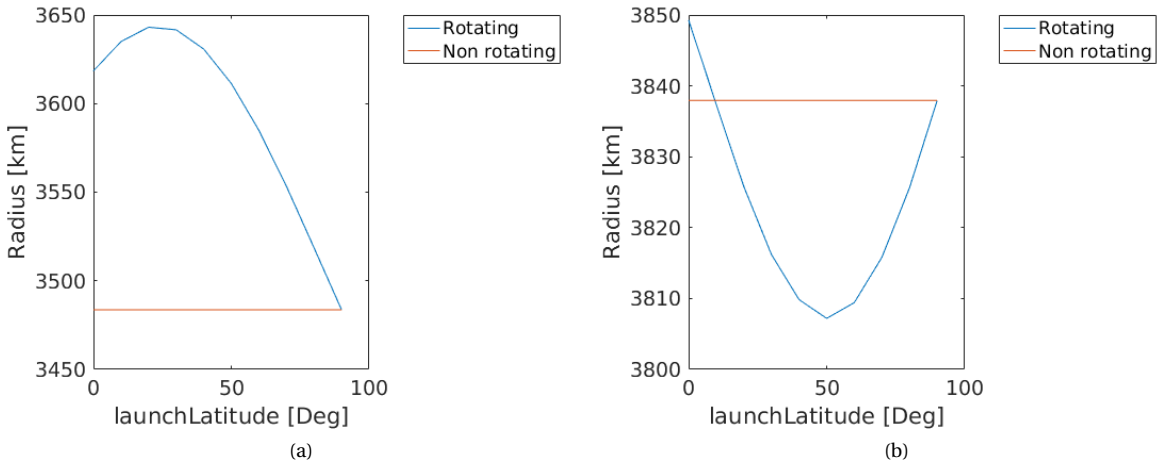


Figure 8.17: Launch latitude versus orbital radius for rotating and non-rotating Mars (a) case 1, (b) case 2

Looking at case 1, the end radius shows a sinusoidal effect with the highest end radius at 20 degrees. It is however interesting that case 2 shows an almost mirror effect, where the maximum end radius is reached

using the nominal conditions, but still a sinusoidal behaviour can be identified. The minimum end radius is reached at 50 degrees and the non-rotating end radius is crossed at a launch latitude of approximately 10 degrees. This sinusoidal effect can only be attributed to the rotating effects which is similar for both case 1 and 2. They still look different though, which is due to the fact that both trajectories are very different. Case 1 launches due East and then North and case 2 launches due East and then South, South-West. Which means that the trajectories are practically mirrored, and this is also seen in the radius graphs.

These effects are also shown in the velocity curves presented in Figure 8.18. Here the ground velocity shows the expected behaviour and a similar curve is also shown by the inertial velocity. However, for case 1 the inertial velocity starts above the non-rotating velocity and then drops below. This transition happens at the maximum radius, after which the curve follows slowly recovers to towards the non-rotating values. At that crossing point, the MAV ends up due West, however this effect is maximum at a latitude of 50 degrees and becomes smaller again as soon as the latitude reaches the North-pole.

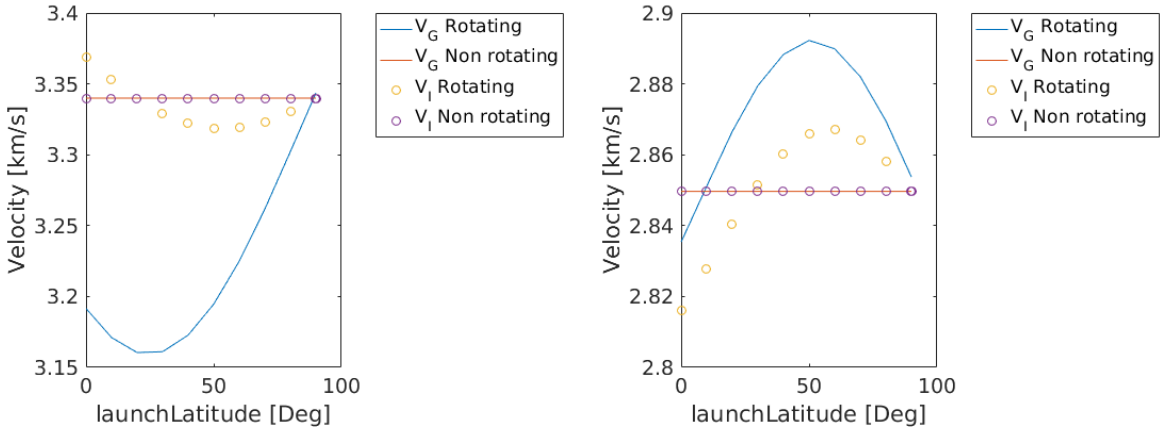


Figure 8.18: Launch latitude versus velocity for rotating and non-rotating Mars (a) case 1, (b) case 2

8.6. LAUNCH LONGITUDE

For the launch longitude, the cut-off criteria was an end time of 789 seconds for case 1 and 873 seconds for case 2. The longitude (τ) has been varied from 0 to 330 degrees. Both the kinematic equations described in Equation (5.66) and the dynamic equations described in Equation (5.67) do not depend on the longitude. This means that no differences are expected between the different longitudes.

In Figure 8.19 the launch longitude is plotted against the CPU time, and even though no two longitudes have the same CPU time there is still no significant difference, because the differences in CPU time can be explained by the arguments presented in Section 8.3.

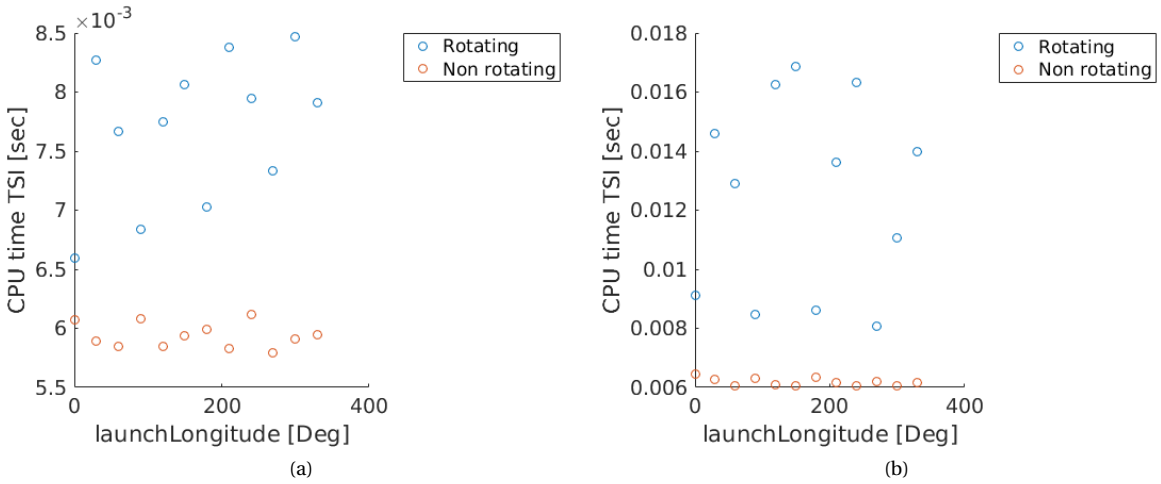


Figure 8.19: Launch longitude versus CPU time for rotating and non-rotating Mars (a) case 1, (b) case 2

It is however more interesting to look at the differences in radius and velocity as shown in Figures 8.20 and 8.21 respectively. Here it can be seen that for both the non-rotating as well as the rotating runs there are no noticeable differences for the end radius and end velocity of case 1. But case 2 does indeed show a difference in both graphs. As a matter of fact depending on the chosen launch longitude, the difference in end radius could be as large as 1 km (a more detailed graph with the differences between the non-rotating and rotating runs can be found in Appendix F, which show that case 1 also has a slight difference but at an order of metres).

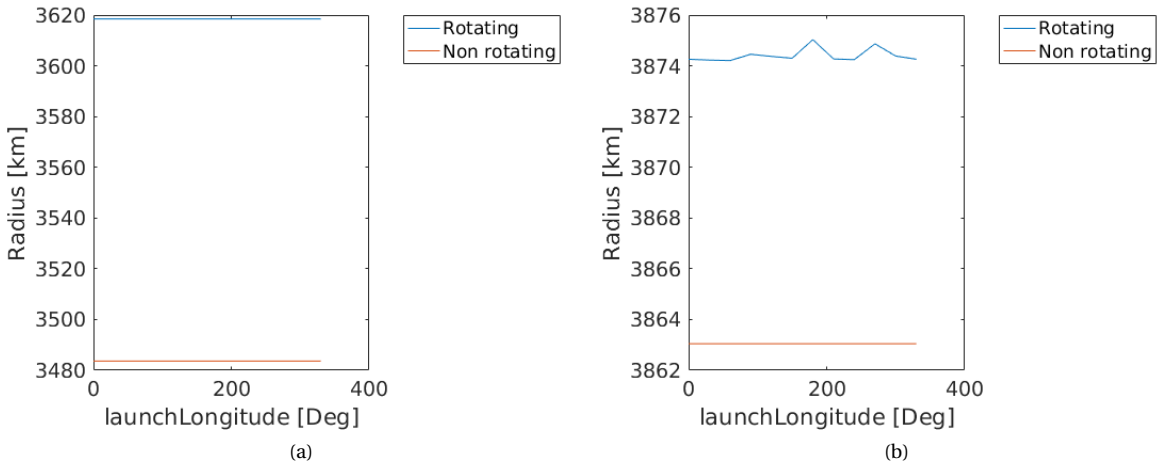


Figure 8.20: Launch longitude versus orbital radius for rotating and non-rotating Mars (a) case 1, (b) case 2

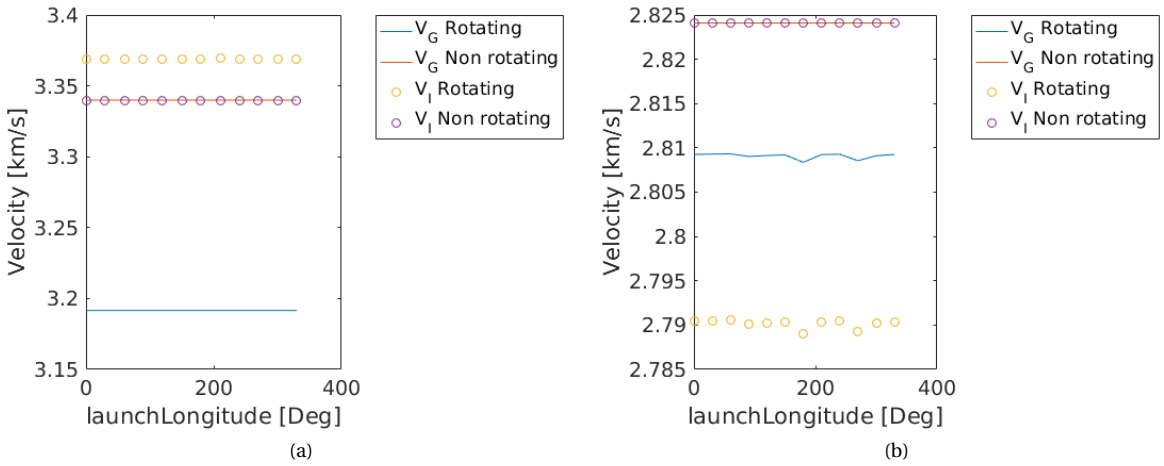


Figure 8.21: Launch longitude versus velocity for rotating and non-rotating Mars (a) case 1, (b) case 2

The difference in end radius follows from a difference in velocity. Because the only difference between case 1 and case 2 is the input, these differences can be attributed to rounding errors. It seems that different combinations of input values could create cases where the rounding errors can have a large effect. It is interesting to note that RKF showed the same deviations.

8.7. FLIGHT-PATH ANGLE

Previously, the cut-off criteria was always set to be a certain end time. However, for the FPA it is better to set the cut-off criteria as the point when the simulation reaches the zero degree FPA. This means that not only the final altitude will be different but also the simulation run time. It is expected that a MAV launched at a lower launch FPA will reach the cut-off criteria earlier compared to a high flight-path angle launch. A lower simulation time will equal fewer function evaluations. This is shown in Figure 8.22, where it is clear that for both the rotating and the non-rotating Mars runs the number of function evaluations decrease with a decrease in FPA for both cases. It also shows that the number of function evaluations does not differ much compared to the non-rotating runs, but the two curves do seem to diverge slightly with higher FPA. This is expected since the simulation run time is higher as well.

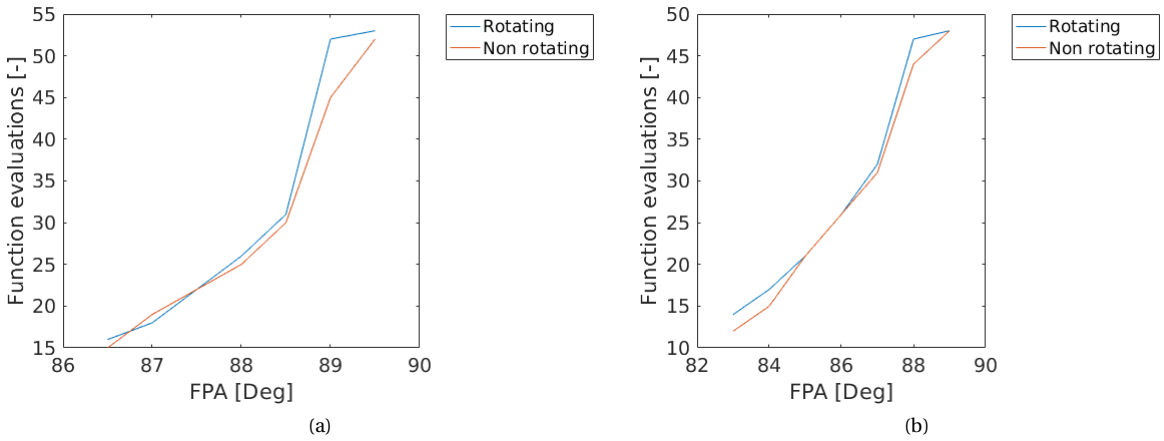


Figure 8.22: Flight-path angle versus Function Evaluations for rotating and non-rotating Mars (a) case 1, (b) case 2

This divergent behaviour can also be spotted in Figure 8.23, where the curve of the non-rotating runs is always below the rotating runs. These graphs also show that there is a clear relationship between the number of function evaluations and the required CPU time.

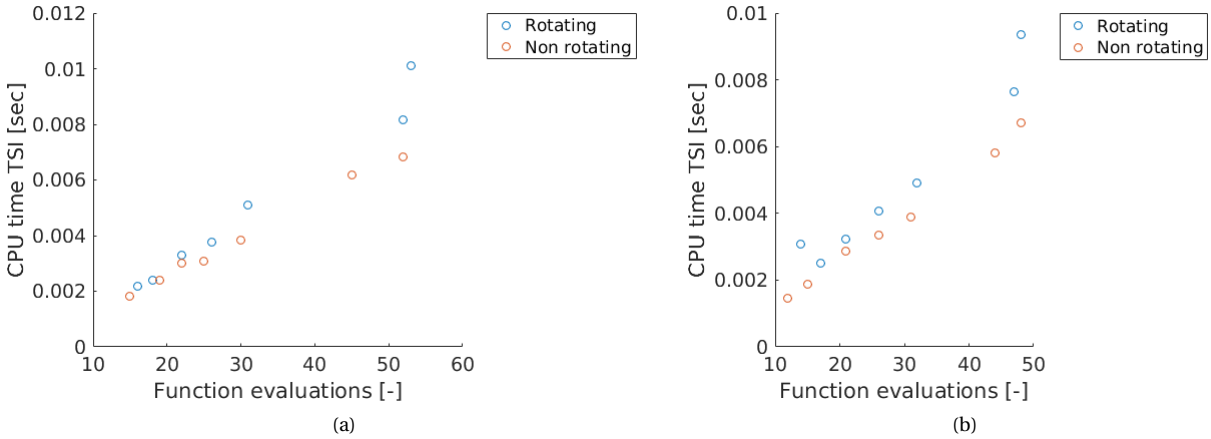


Figure 8.23: Function evaluations versus CPU time for rotating and non-rotating Mars (a) case 1, (b) case 2

As was mentioned in the introduction of this section, the MAV launching at a lower FPA is expected to reach a lower altitude because the zero degree cut-off criteria is reached earlier in time. This is precisely what is shown in Figure 8.24 where the lower FPA's only reach an effective altitude of a few metres, whereas the higher FPA's are able to reached an altitude of a few hundred kilometres. This is also due to the fact that besides the FPA nothing else changed in between runs, which in turn means that every run uses the same thrust profile. The nominal thrust profile was set for a FPA of 89 degrees. If the MAV is then set at a much shallower angle at launch it means that this thrust is directed in a more horizontal direction (in comparison) and will therefore have a much larger effect on the horizontal flight profile.

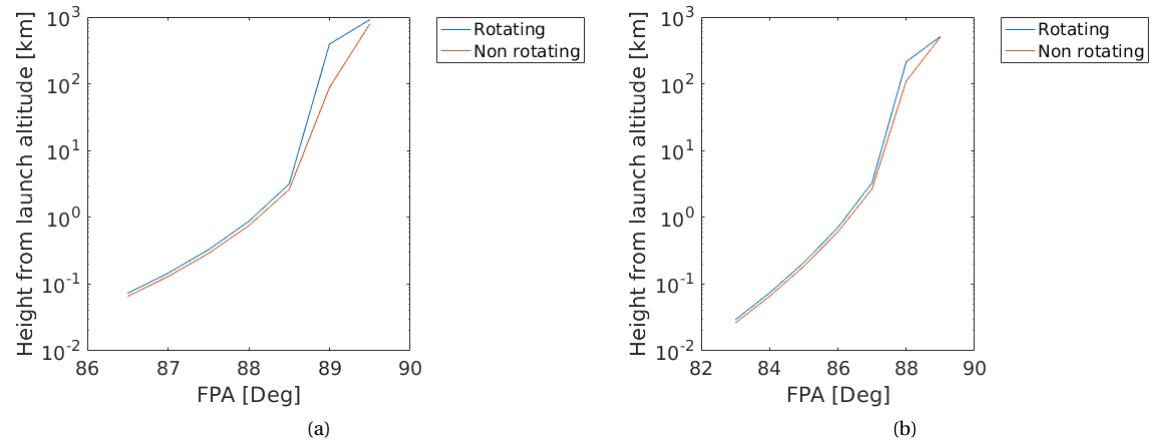


Figure 8.24: Flight-path angle versus end height from launch site for rotating and non-rotating Mars (a) case 1, (b) case 2

Because each of the runs is cut-off at the zero FPA, it is interesting to look at the required propellant mass to reach a circular orbit with the desired inclination at that particular point. Please note that these circularisation burns are assumed to be impulsive and thus only results in an approximation of the required propellant mass. In Figure 8.25 the non-rotating Mars curve is situated above the rotating Mars curve. This indicates that, even though it might not be much, the non-rotating Mars case requires more propellant mass to circularise the orbit due to the fact that it is lacking the initial boost of the rotating Mars.

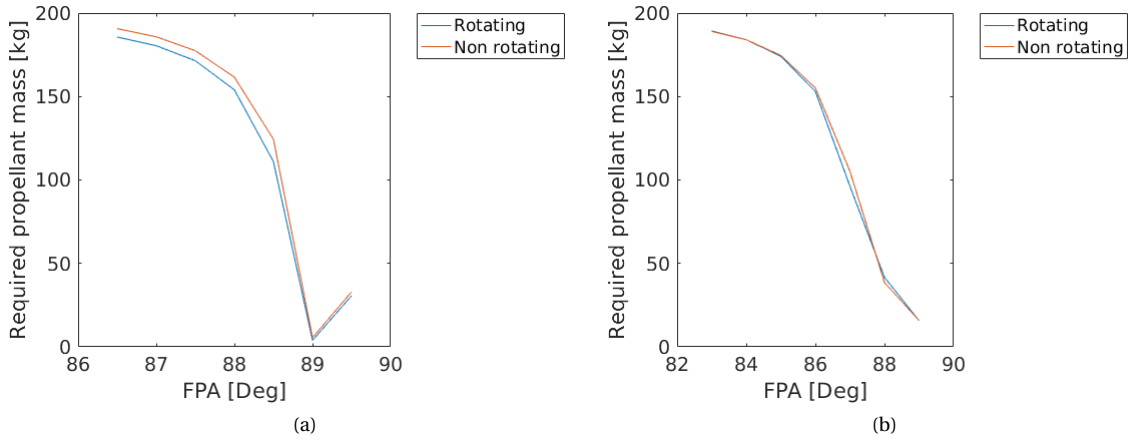


Figure 8.25: Flight-path angle versus propellant mass required for circularisation and desired inclination change for rotating and non-rotating Mars (a) case 1, (b) case 2

For case 1 the FPA range was from 86.5 till 89.5 degrees, and for case 2 the range ws from 83 till 89 degrees. For both cases the minimum required propellant mass to circularise is at a FPA of 89 degrees, which is exactly the nominal case for which the thrust profile was calibrated. It should also be said that the atmospheric effects have also not been taken into account for the calculation of the required propellant mass for the circularisation. This is therefore purely theoretical, because a circularisation at an altitude of a few metres would mean that the MAV would hit a mountain or some other elevated area at some point. But it does show the large effect that the launch FPA has on the trajectory.

8.8. HEADING ANGLE

The final variable that is investigated is the launch heading angle. This angle defines the direction in which the MAV is pointed at launch. In this thesis research the heading angle is defined to between 0 degrees facing North, 90 degrees due East, 180 degrees pointed South and 270 degrees due West. Again the cut-off criteria for this angle has been set at a FPA of 0 degrees. Thus the end time and altitude are again left free. This means that the number of function evaluations will also be different as shown in Figure 8.26.

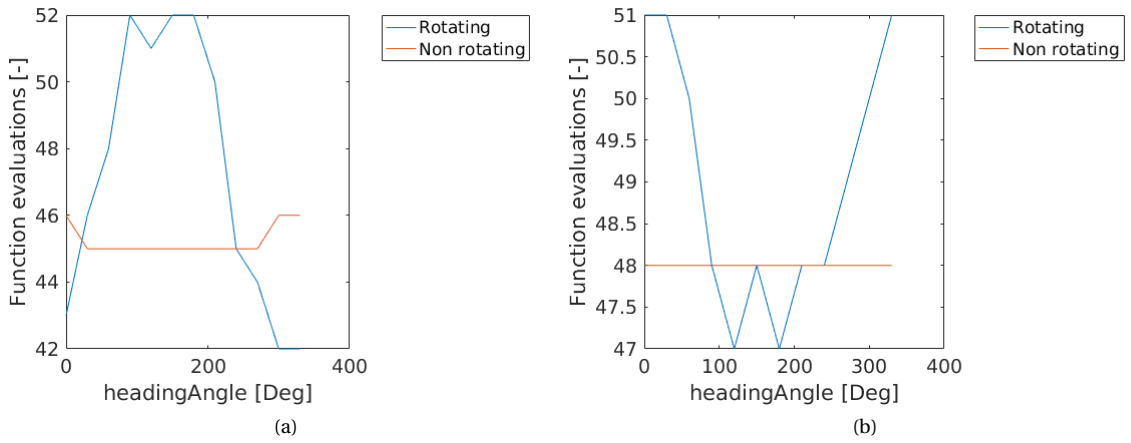


Figure 8.26: Heading angle versus Function Evaluations for rotating and non-rotating Mars (a) case 1, (b) case 2

For case 2 it appears that the heading angle does not have any influence on the number of function evaluations required. This is expected since the rotation of Mars is not included which negates any arbitrary rotational influences. Therefore, the zero degree FPA will be reached at the same simulation time. However, case 1 shows that sometimes the number of function evaluations is 46 (between 300 and 0 degrees) and some-

times it is 45. That could only happen if the values are close to the boundary values of the allowed precision which determines the step-size. Therefore, an end time difference of 1 μs can result in having to perform one more simulation step.

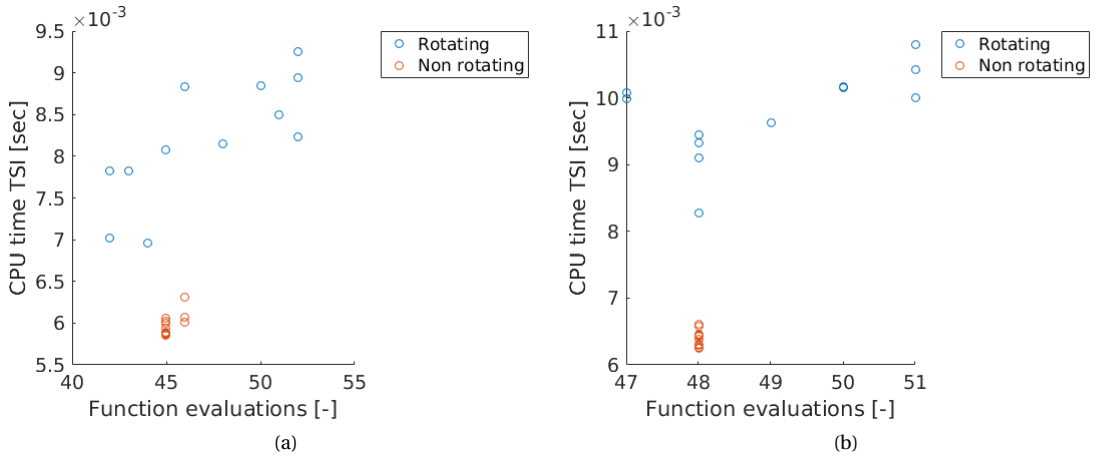


Figure 8.27: Function evaluations versus CPU time for rotating and non-rotating Mars (a) case 1, (b) case 2

In Figure 8.28 the function evaluations are again plotted against the CPU time and here the same trend can be observed as was seen for the FPA results.

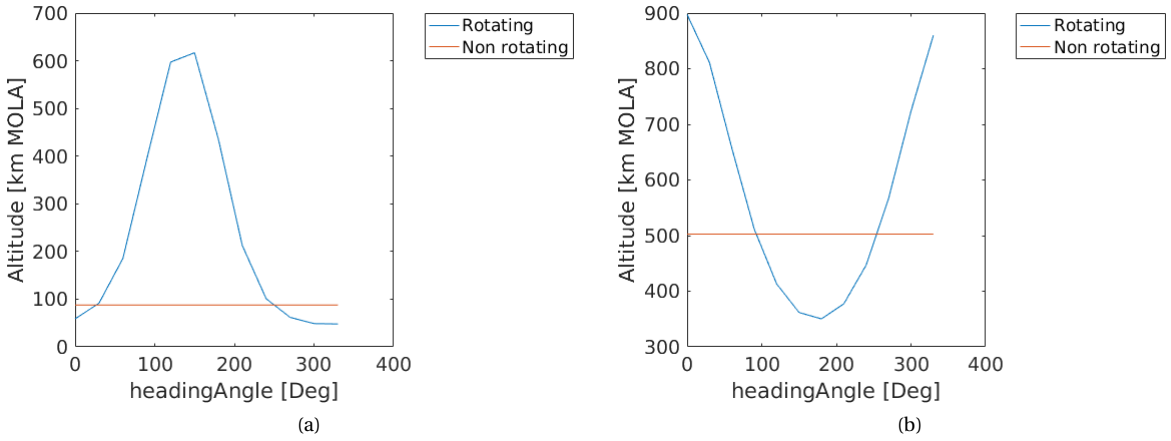


Figure 8.28: Heading angle versus altitude with respect to the MOLA reference for rotating and non-rotating Mars (a) case 1, (b) case 2

Because only the heading angle changes but the rest of the variables remain the same, it would be expected that the reached altitude of the MAV would show a sinusoidal behaviour for both cases. When inspecting Figure 8.29 this behaviour is indeed present in both cases. In fact, the curve of case 1 could effectively be moved as to create the curve at case 2. The non-rotating curves are both crossed due to the fact that the trajectory for both cases result in a flight path that heads due West in the end. It is interesting to note that the maximum altitude that can be reached lies higher in both cases than the nominal case of 90 degrees.

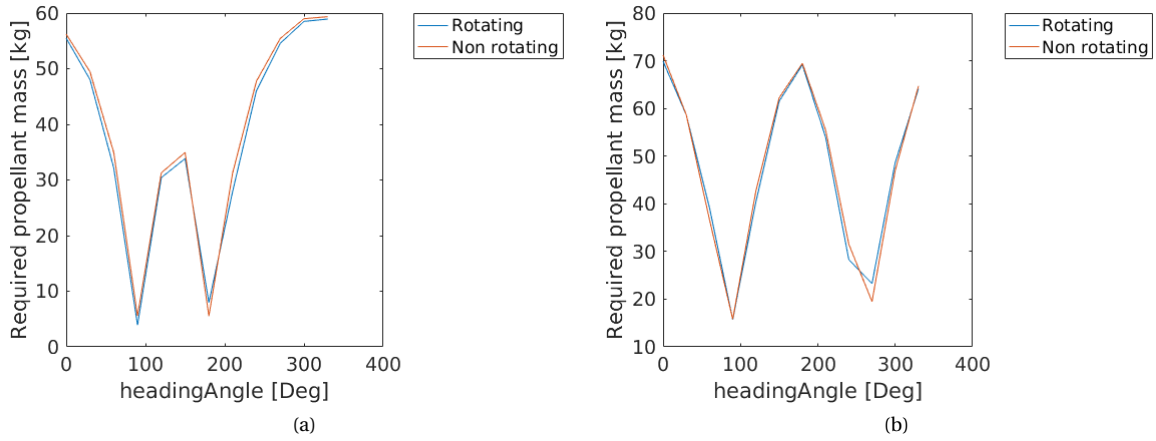


Figure 8.29: Heading angle versus propellant mass required for circularisation and desired inclination change for rotating and non-rotating Mars (a) case 1, (b) case 2

The reached altitude might be higher, but the propellant mass required to reach the desired orbit inclination and circularise the orbit at that altitude might be very high as well. In fact, Figure 8.30 shows that there are only two local minima when it comes to propellant use. And one of these local minima is the global minimum at 90 degrees for which the nominal case was adjusted. These second minima were however a bit unexpected at first. For the first case it lies at 180 degrees and the second case at 270 degrees. When taking a closer look it becomes clear that these are the cases where the MAV is able to get into the same orbit but then passing over the equator from North to South in case 1 and from South to North in case 2. Effectively thus not requiring much energy for the change in inclination.

This same behaviour can be seen in the inertial velocity curves for both cases shown in Figure 8.30.

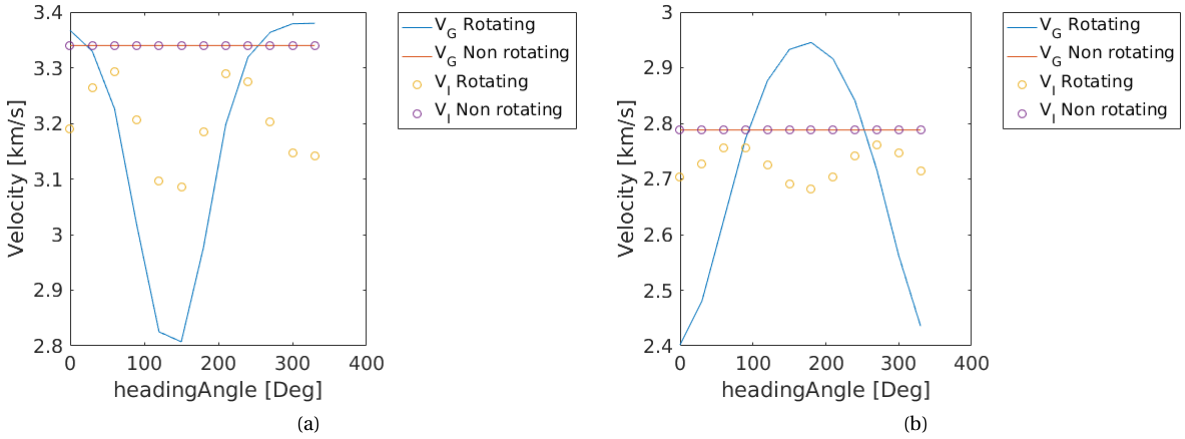


Figure 8.30: Heading angle versus velocity for rotating and non-rotating Mars (a) case 1, (b) case 2

Here the ground velocity and non-rotating velocities are an exact mirror of the radius curves and the inertial velocity curve shows two maxima. These velocity maxima do not however correspond exactly to the optimal propellant masses found. This is because the inclination change also requires some propellant.

9

CONCLUSIONS AND RECOMMENDATIONS

During the course of this research, many different aspects of TSI have been investigated. With each phase new conclusions were drawn, even during the programming and setting up the integrator. All these conclusions are described in Section 9.1. However, which each conclusion also came a sense of what could still be improved and added. All the recommendations that came out of that process are described in Section 9.2.

9.1. CONCLUSIONS

The conclusions can be divided up into three different parts. The first part are the conclusions drawn as the TSI method was worked out and programmed. During this phase, some of the advantages and disadvantages of this method became clear and are discussed in Section 9.1.1. Then one of the main aspects of this research is described in Section 9.1.2, where the conclusions are presented with respect to the comparison between TSI and RKF. And finally, a sensitivity analysis was performed on TSI using some of the different variables that the program requires as input. The conclusions from this analysis are presented in Section 9.1.3.

9.1.1. TAYLOR SERIES AS AN INTEGRATOR

Setting up the TSI turned out to be more difficult than anticipated, because it requires all the written out equations for the model. These include the equations to simulate the atmosphere and equations to compute the drag coefficient. All the equations involved also require the first derivatives, and each of these derivatives has to be divided up into auxiliary functions. In this process of setting up all these equations, it is really easy to make mistakes. In this theses, the basic recurrence relations were simply programmed into a single class such that each of the recurrence relations for the auxiliary derivatives and functions could call those basic functions. But in the end it still took a long time to set up. Because the auxiliary derivatives and functions are intertwined, adding an extra perturbation for example could be as simple as adding a few more equations, or as difficult as rewriting some of these equations. It is therefore easier to include as many perturbations when designing the integrator such that during the actual simulation one can simply specify the desired perturbation that have to be taken into account. This would make the program itself less problem specific.

Once the program was all set-up, a step-size handling procedure had to be included to deal with the different sections in the temperature and drag coefficient model. This was needed to make sure that TSI did not continue into the next section without changing the section properties. The step-size handling procedure was easily implemented and is able to determine when a next section is reached and change the step-size such that the previous step ends at the start of the next section. The atmospheric and drag coefficient models were set-up at the start of the thesis research and at that time a certain number of sections were fitted as to get the best representation of the corresponding model. However, during the development of TSI it became clear that it would be better to have a model with fewer sections. The reason for having more than 1 sections was because this limited the order of the polynomial that had to be fitted. But every time a new section is reached, TSI has to limit a step-size to pass over that boundary. Therefore it would be better to have large order polynomial fits and fewer sections. Fortunately, this model change can effortlessly be implemented in the simulation program by simply updating the polynomial coefficient matrices, which are an input of the integrator. Because TSI has this ability to readily update the step-size to meet a certain section boundary

condition, it means that it can also freely cope with discontinuities.
Another advantage is that TSI can theoretically have any desired order. This theoretically unlimited order is however limited by the computer accuracy, because at a certain point the higher orders disappear into the rounding errors.
And finally, the Taylor Series coefficients are computed for each time step after which a corresponding step-size is computed to stay within a certain error tolerance. The coefficients are then multiplied by the step-size as per Equation (5.102). This step-size is the maximum step-size that can be used to determine the next state while still staying within the error tolerance, however if a smaller step-size is chosen, it means that any value between the current time step and the next time step can be computed as well (meeting an even higher error tolerance). The current program has the option to save all the Taylor Series coefficients for each time step, which means that even after the simulation has been performed extra data points can be created in between the already chosen time steps by simply choosing a smaller time step as mentioned before. Interpolation is therefore not required and the data can have a theoretically unlimited resolution, which is a major advantage over conventional integrators.

9.1.2. TSI COMPARED TO RKF78

The main question that had to be answered in this research was: "Does TSI show similar performance improvements over RKF for an ascent case as was shown for orbital trajectories and (re-)entry cases?". There are a few different aspects that were compared between TSI and RKF78.
First, the number of function calls were compared. In this thesis research, a function call is the process of going through the set of equations once. For RKF78 this has to be done 13 times in one time step, whereas it only has to be done once per time step for TSI. The effect of this difference was made very clear when both TSI and RKF were run for a number of different error tolerances. Each error tolerance resulted in a different number of time steps in order to stay within the error bounds. For all tested error tolerances, TSI required far fewer function calls than RKF. Not only that, but as the error tolerance increased, the number of function calls required for RKF increased rather quickly as well. This resulted in a maximum number of evaluations required for RKF to be 7 to 8 times more than for the lowest error tolerance. This compared to TSI where the difference in number of evaluations between the lowest and the highest error tolerance was less than twice the lowest number of evaluations.
The second aspect that was compared was the accuracy. Again using the error tolerance, the accuracy of both methods at different error tolerances could be determined. For TSI the accuracy of the position and velocity at an error tolerance of 10^{-5} was 1 cm and 10 μm respectively, whereas RKF was only able to reach these accuracies at an error tolerance of 10^{-9}. This clearly shows that TSI is inherently more accurate even at lower error tolerances. Besides being more accurate, TSI also showed a faster convergence speed compared to RKF, which meant that the highest tolerances did not show much of a difference.
Finally, the CPU time of both methods were compared as well. CPU time is measured by the time it takes the CPU to run through all the computations required for the simulation. In all tested cases TSI turned out to be a whole order slower, on average, than RKF. This is the complete opposite of what was expected based on previous research. The RKF78 method was provided by the Tudat library and only required a single state derivative file in order to work. TSI on the other hand was programmed in such a manner that is was comprehensible, but in doing so many different program files were created that all had to be called during the integration. Each of those function files contains computations that could be run at the same time, but instead they were run consecutively. Therefore, the CPU time could be decreased by using different threads. One example where this could have a large impact is the computation of the Taylor Series coefficients. Not all the coefficients are interdependent and could therefore be run in separate threads at the same time. The actual improvement on CPU time that this threading method would have is not yet clear but is definitely something that should be investigated.

9.1.3. SENSITIVITY ANALYSIS

To be able to understand the behaviour of TSI better and to determine the influence of different parameters on the ascent trajectory, a sensitivity analysis was performed. In this section, the conclusions will be presented per parameter starting with the TSI order.

ORDER

During the different order runs for both cases it was determined that an order of 20 resulted in the lowest CPU time on average. There were also two different phases identified; one before and one after this lowest CPU

time order. In the first phase, the CPU time was greatly influenced by the high number of time steps that were needed because of the inaccuracies introduced by the lower orders. The second phase showed that the CPU time was now influenced more by the number of recurrence relation computations that were introduced with each extra order. Also, the number of time steps decreased only a little between the first order of that phase and the final order of that phase and thus reduced the effect of the number of time steps on the CPU time. Finally, it was also shown that from an order of 11 the accuracy of the method did not change that much any more compared to the nominal RKF run. But since order 20 was still faster, it was better to run the simulation at that order.

ERROR TOLERANCE

As was already mentioned in Section 9.1.2, the number of evaluations for TSI varied very little between the different error tolerances. In both cases an error tolerance of 10^{-5} resulted in 24 evaluations (in that particular simulation run) and an error tolerance of 10^{-15} resulted in 46 and 47 evaluations for case 1 and 2 respectively. On average this was an increase of 2.2 and 2.3 respective evaluations per error tolerance. Because the error tolerance has a direct correlation to the number of time steps, it also had a direct influence on the CPU time. An increase in error tolerance therefore resulted in an increase in CPU time. Also, the non-rotating Mars runs were approximately 0.2 ms faster compared to the rotating Mars runs, which was to be expected since fewer equations had to be performed.
It was also found that a lower tolerance resulted in a less accurate final result compared to the higher tolerance results, which makes sense since a lower error tolerance means that a larger error is accepted making the end results less accurate.

MULTIPLE RUNS

The multiple nominal runs showed the influence of computer processes on the CPU time of the integration run. This influence of processes running on the background sometimes resulted in a difference in CPU time of 30% or more, even though the exact same case was run. This influence was best shown during a rung of the non-rotating Mars scenario where a sudden spike appeared in the CPU data, indicating that another process was active at that time in the background. This does not explain the difference between the RKF and TSI CPU times, but it does show that CPU time is not always a good comparison parameter.

LAUNCH ALTITUDE

Keeping in mind that a certain variation in CPU times is allowed due to the behaviour observed during the multiple runs, no direct influences of launch altitude on the CPU time could be identified. The launch altitude did however have a direct influence on both the final radius as well as the final velocity of the MAV. An increase in launch altitude resulted in an increase in final radius and a decrease in the final velocity.

LAUNCH LATITUDE

The CPU time of the launch latitude runs showed an undeniable influence of the different latitudes. Starting at the equator and ending up at the North-pole, the CPU time showed a convergence behaviour of the rotating and the non-rotating Mars curves. Therefore, the closer the MAV launches to the North-pole, the smaller the influence of the rotation. The final radius also highly depends on the chosen launch latitude. Here the radius curve showed a sinusoidal behaviour in relation to the launch latitude, and a same correlation was found in the velocity curves. In the non-rotating Mars runs, the latitude had absolutely no influence on the final radius and velocity what so ever. This means that the chosen launch latitude has a large effect on the final radius and velocity due to the rotation of Mars.

LAUNCH LONGITUDE

Compared to the launch latitude, the launch longitude has virtually no influence at all on the final radius and velocity. This is due to the fact that the rotational influences do not change in the longitudinal direction.

FLIGHT-PATH ANGLE

When running the different FPA cases it became clear that the FPA has a large influence on the flight trajectory of the MAV. For these runs the cut-off criteria was a zero degree FPA. With a decrease in launch FPA that cut-off criteria was reached earlier in flight each time. Up until the point where a difference of 2 degrees could mean reaching the desired orbit or not even reaching an altitude of 1 km. For these runs, the orbit was also circularised (which included an inclination change to the desired inclination) once the cut-off criteria was

met. This showed a massive difference in the required propellant mass required, because the required orbital velocity became more and more difficult to reach the lower the launch FPA and because the flight was cut-off earlier, the MAV did not have enough time to change its heading to match the desired inclination which also added to the required propellant mass.

HEADING ANGLE

Because the other parameters were all left constant in between the launch heading angle runs, a sinusoidal behaviour was observed in the altitude versus launch heading angle curve. And it was interesting to see that for both case 1 and 2 the maximum reachable altitude with those parameters was not at the nominal 90 degree angle. However, these maximum reachable altitudes also required a lot of propellant mass to get into the desired orbit with the desired inclination. Since the nominal case was calibrated for the desired orbit it was expected to required the least amount of propellant mass to reach that orbit, and this indeed turned out to be the case. However, another local minimum propellant option was identified for both case 1 and case 2. They required slightly more propellant but were still a convincing option. These two local minima are the cases where the exact same orbit is reached but then moving in the opposite direction over the equator compared to the nominal trajectory. It was interesting to see that a change in launch heading angle could identify this second local minimum, which also clearly shows that the launch heading angle has a large effect on the launch trajectory.

9.2. RECOMMENDATIONS

The recommendations are split into two categories: recommendations to improve the current TSI program (Section 9.2.1) and recommended tests and additions that could be made (Section 9.2.2). The improvements to the current TSI program are needed to try to reduce the CPU time and see if the same performance can be achieved as shown by Scott and Martini (2008); Bergsma and Mooij (2016) and also to try and make the TSI program more robust. The extra tests and additions are either planned processes that could not be executed in the scope of this thesis or processes that were not originally planned but would still be interesting to investigate.

9.2.1. IMPROVEMENTS TO THE CURRENT TSI PROGRAM

The list presented in this section suggests some steps that could be taken to maybe improve the performance of the current TSI program.

Use threads to perform simultaneous computations The current program does not use any threads at all, instead the computations are all run one after another. However, there are many computations that could be performed in parallel because they are independent. Using threads to perform these simultaneous computations would theoretically reduce the CPU time.

Get rid of obsolete computations and functions Even though most of the obsolete computations and functions were removed during the many rewrites of the program, it could be that some of the old code is still present but not used any more. Removing the obsolete code could theoretically reduce the CPU time.

Get rid of the separate basic recurrence relations file In the current program, the basic recurrence relations are functions that are written in a separate file and called by the final recurrence relations file. In the reference research, the basic recurrence relations are always described completely worked out in the final recurrence relations. The improvements on the program by getting rid of the basic recurrence relations file and writing them out in the final recurrence file are yet unknown.

Use a different root finding method The current root finding method was based on the root finding method used by Bergsma and Mooij (2016), however it could be that another root finding method would work better for the tested scenarios.

Implement a different step-size determination method The step-size determination method worked well for the current program, but it showed some divergence issues when directly applied to the starting conditions. This is why an initial second was integrated using RKF78. A different step-size determination method might be able to better estimate the required step-size, although it is unlikely to completely get rid of the early divergence issues in the Taylor Series coefficients.

Use a temperature model with fewer sections but higher order polynomial fits Because TSI had to stop with the current time step and adjust the step-size accordingly every time a new section was reached, it would be better to have use a temperature model with fewer sections. This does mean that functions fits with higher order polynomials are required, but TSI does not seem to have an issue with those. The same goes for the drag coefficient model.

9.2.2. EXTRA TESTS AND ADDITIONS

The tests and additions to the program listed here are suggestions based on the experience gathered during this research and academic interest.

Add optimisation Optimisation is something that still needs to be implemented and tested. Use can be made of the **Pagmo!** (**Pagmo!**) library in combination with the Sparse Nonlinear Optimizer (SNOPT) method. Unfortunately, this could not be done during this research.

Add extra perturbing accelerations Even though other perturbing effects such as higher order gravity terms or third body perturbations could be neglected, it would still be interesting to see what the influence on the trajectory would be.

Test the influence of the drag, thrust and gravitational acceleration A more extensive sensitivity analysis could be performed where different scenarios could be tested that include a thrust profile instead of a constant nominal thrust, the actual influence of the drag on the trajectory and what would happen for instance if the gravitational acceleration would be different.

Include the thrust angle profile For the complete trajectory analysis and comparison to the numerical analysis currently being performed at JPL it would be interesting to include the thrust angle profile. This could be done in combination with the optimisation to determine the required thrust profile for the optimum trajectory.

BIBLIOGRAPHY

R. Shotwell, J. Benito, A. Karp, and J. Dankanich, *Drivers, developments and options under consideration for a Mars ascent vehicle,* in *Aerospace Conference, 2016 IEEE* (IEEE, 2016) pp. 1–14.

R. C. Woolley, R. L. Mattingly, J. E. Riedel, and E. J. Sturm, *Mars Sample Return - Launch and Detection Strategies for Orbital Rendezvous,* in *AAS/AIAA Astrodynamics Specialist Conference*, Vol. 142 (Univelt, Inc., 2011).

D. Vaughan, B. Nakazono, A. Karp, R. Shotwell, A. London, A. Mehra, and F. Mechentel, *Technology development and design of liquid bi-propellant mars ascent vehicles,* in *Aerospace Conference, 2016 IEEE* (IEEE, 2016) pp. 1–12.

J. Fanning and B. Pierson, *A model comparison for optimal Mars ascent trajectories,* in *Engineering Optimization*, Vol. 26 (1996) pp. 271–285.

P. Desai, R. Braun, W. Engelund, F. Cheatwood, and J. Kangas, *Mars Ascent Vehicle Flight Analysis,* in *7th AIAA/ASME Joint Thermophysics and Heat Transfer Conference* (1998).

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).

J. C. Whitehead, *Trajectory analysis and staging trades for smaller Mars ascent vehicles,* in *Journal of spacecraft and rockets*, Vol. 42 (2005) pp. 1039–1046.

E. Di Sotto, J. C. Bastante, and R. Drai, *System and GNC concept for RendezVous into elliptical Orbit for Mars Sample Return mission,* in *AIAA Guidance, Navigation, and Control Conference and Exhibit, Colorado* (2007).

M. Trinidad, E. Zabrensky, and A. Sengupta, *Mars Ascent Vehicle system studies and baseline conceptual design,* in *Aerospace Conference, 2012 IEEE* (IEEE, 2012) pp. 1–13.

E. Dumont, *Design of a Modular Transportation System for Future Lunar Robotic Missions,* in *30th ISTS/34th IEPC/6th NSAT Joint Conference, Kobe, Japan* (2015).

R. C. Woolley, *A simple analytic model for estimating Mars Ascent Vehicle mass and performance,* in *2015 IEEE Aerospace Conference* (2015).

J. Benito and B. J. Johnson, *Trajectory Optimization for a Mars Ascent Vehicle,* in *AIAA/AAS Astrodynamics SPecialist Conference* (2016).

O. Montenbruck, *Numerical integration of orbital motion using Taylor series,* in *Spaceflight mechanics* (1992) pp. 1217–1231, available at JPL library.

J. R. Scott and M. C. Martini, *High speed solution of spacecraft trajectory problems using Taylor series integration,* in *AIAA/AAS Astrodynamics Specialist Conference and Exhibit, Honolulu, Hawaii* (2008).

M. Bergsma and E. Mooij, *Application of Taylor-Series Integration to Reentry Problems with Wind,* in *Journal of Guidance, Control, and Dynamics* (American Institute of Aeronautics and Astronautics, 2016) pp. 2324–2335.

J. Whitehead, *Mars ascent propulsion options for small sample return vehicles,* in *33rd AIAA/ASME/SAE/ASEE Joint Propulsion Conference and Exhibit* (AIAA Joint Propulsion Conference and Exhibit, 1997).

C. S. Guernsey, *Mars Ascent Propulsion System (MAPS) Technology Program: Plans and Progress,* in *34th AIAA/ASME/SAE/ASEE Joint Propulsion Conference and Exhibit* (1998).

C. Stone, *Rice Mars,* presentation/website (1999), [online database], URL: http://www.slideshare.net/cliffordstone/rice-mars-nov99 [cited 25 October 2015].

D. Stephenson, *Mars ascent vehicle - Concept development,* in *38th Joint Propulsion Conference and Exhibit, Indianapolos, Indiana*, Vol. 4318 (AIAA, 2002).

D. D. Stephenson and H. J. Willenberg, *Mars ascent vehicle key elements of a Mars Sample Return mission,* in *Aerospace Conference, 2006 IEEE* (IEEE, 2006) p. 11.

A. Sengupta, A. Kennett, M. Pauken, M. Trinidad, and E. Zabrensky, *Systems Engineering and Technology Considerations of a Mars Ascent Vehicle,* in *2011 IEEE Aerospace Conference* (2012).

G. Mungas, D. Fisher, J. Vozoff, and M. Villa, *NOFBX Single Stage to Orbit Mars Ascent Vehicle,* in *Aerospace Conference, 2012 IEEE* (IEEE, 2012) pp. 1–11.

Mars Program Planning Group, *Summary of the Final Report,* (2012), [online database], URL: http://www.nasa.gov//sites/default/files/files/ [cited 14 October 2015].

A. C. Karp, M. Redmond, B. Nakazono, D. Vaughan, R. Shotwell, G. Story, D. Jackson, and D. Young, *Technology development and design of a hybrid Mars ascent vehicle concept,* in *Aerospace Conference, 2016 IEEE* (IEEE, 2016) pp. 1–10.

R. Shotwell, *History of Mars Ascent Vehicle development over the last 20 years,* in *Aerospace Conference, 2016 IEEE* (IEEE, 2016) pp. 1–11.

J. Mulder, W. van Staveren, J. van der Vaart, E. de Weerdt, A. in 't Veld, and E. Mooij, *Flight Dynamics, Lecture Notes,* Reader, Delft University of Technology (2013), [Internal publication] Course: Flight Dynamics.

E. Mooij, *The motion of a vehicle in a planetary atmosphere* (Delft University Press, Delft, 1994).

E. Mooij, *AE3202-week1-lecture1b-Frames,* Lecture slides, Delft University of Technology (2013), [Internal publication] Course: Flight Dynamics.

R. Noomen, *ae4-878.basics.v4-14,* Lecture slides, Delft University of Technology (2013a), [Internal publication] Course: Mission Geometry and Orbit Design.

C. Ho, N. Golshan, and A. Kliore, *Radio Wave Propagation Handbook for Communication on and Around Mars*, 1st ed. (NASA Jet Propulsion Laboratory, Pasadena, CA, 2002) handbook JPL Publication 02-5.

D. Williams, *Mars Fact Sheet,* website (2015), [online database], URL: http://nssdc.gsfc.nasa.gov/planetary/factsheet/marsfact.h [cited 1 November 2015].

K. Wakker, *Astrodynamics-II Course AE4874, part 2,* Reader, Delft University of Technology (2010), [Internal publication] Course: Astrodynamics-II.

R. Hofsteenge, *Computational Methods for the Long-Term Propagation of Space Debris Orbits*, Master's thesis, Delft University of Technology (2013).

A. Milani and A. M. Nobili, *Integration error over very long time spans,* in *Celestial mechanics*, Vol. 43 (Springer, 1987) pp. 1–34.

R. Noomen, *ae4-878.integrators.v4-3,* Lecture slides, Delft University of Technology (2013b), [Internal publication] Course: Mission Geometry and Orbit Design.

P. Deuflhard, U. Nowak, and U. Poehle, *Program Descriptions of ELib,* (1994), [online database], URL: http://elib.zib.de/pub/elib/codelib/difex2/readme [cited 30 October 2015].

O. Montenbruck, *Numerical integration methods for orbital motion,* in *Celestial Mechanics and Dynamical Astronomy*, Vol. 53 (1992) pp. 59–69.

J. Dormand, M. El-Mikkawy, and P. Prince, *Families of Runge-Kutta-Nystrom formulae,* in *IMA Journal of Numerical Analysis*, Vol. 7 (1987) pp. 235–250.

E. Fehlberg, *Low order classical Runge-Kutta formulas with stepwise control*, Tech. Rep. (Marschall Space Flight Center, NASA, 1969) : NASA TR R-316.

E. Fehlberg, *Classical fifth-, sixth-, seventh-, and eighth-order Runge-Kutta formulas with stepsize control*, Tech. Rep. (Marschall Space Flight Center, NASA, 1968) : NASA TR R-287.

M. M. Berry, *A Variable-Step Double-Integration Multi-Step Integrator*, Ph.D. thesis, Virginia Polytechnic Institute and State University (2004).

J. Meijaard, *A Comparison of Numerical Integration Methods with a View to Fast Simulation of Mechanical Dynamical Systems,* in *Real-Time Integration Methods for Mechanical System Simulation*, Vol. 69 (Springer, 1991) pp. 329–343.

H. Ramos and J. Vigo-Aguiar, *Variable stepsize Störmer-Cowell methods,* in *Mathematical and Computer Modelling*, Vol. 42 (2005) pp. 837–846.

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

P. Prince and J. Dormand, *High order embedded Runge-Kutta formulae,* in *Journal of Computational and Applied Mathematics*, Vol. 7 (Elsevier, 1981) pp. 67–75.

M. Weeks and S. Thrasher, *Comparison of Fixed and Variable Time Step Trajectory Integration Methods for Cislunar Trajectories,* in *AAS/AIAA Space Flight Mechanics Meeting* (2007).

M. C. W. Bergsma, *Application of Taylor Series Integration to Reentry Problems*, Master's thesis, Delft University of Technology (2015).

A. Jorba and M. Zou, *A Software Package for the Numerical Integration of ODEs by Means of High-Order Taylor Methods,* in *Experimental Mathematics*, Vol. 14 (Taylor & Francis, 2005) pp. 99–117.

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].

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
```

B

ATMOSPHERIC DATA FITTING

In Section 3.4.1 the final results of the atmospheric data fits were presented. However, before those results could be obtained, a few iterations of trial-and-error fits were required. These intermediate trials are presented for both the temperature data, in Appendix B.1, as well as for the density data, in Appendix B.2. The fit requirements for the polynomial function were set such that the errors created by the fit were less than the uncertainty of the latitude and longitude dependent data. This way the desired accuracy could still be reached. This uncertainty is caused by the difference in atmospheric data curves for the different latitudes and longitudes. The atmospheric data curve for the latitude and longitude corresponding to the launch site (21.0 °N and 74.5 °E) was taken as the reference curve and called the launch site curve. The differences between this launch site curve and the 8 other curves were used to define the maximum data curves difference. The *polyfit.m* function within Matlab was used to fit a polynomial to the data. This function also directly provides a standard deviation of the fit for each data point (in combination with the *polyval.m* function). One of the requirements was for the maximum standard deviation of the polynomial fit to be one order less than the maximum difference of the data curves with respect to the launch site curve. The second requirements for a proper fit was for the absolute maximum difference between the polynomial fit and the launch site curve to be less than the absolute maximum difference between the data curves and that same launch site curve.

B.1. TEMPERATURE

Initially, the temperature data curve was split into 6 different sections as portrayed in Figure B.1. The sections were selected on the basis of their individual shapes and the maximum order that was required, where the maximum order for temperature was set at 8. Fewer sections were however always preferred.

Unfortunately, the fourth quarter of section 3 caused this section to not meet the maximum standard deviation requirement because in that quarter the linear behaviour changes directions slightly and the differences between the different data curves become a lot smaller. Therefore section 3 was split into two different sections resulting in a total of 7 sections as shown in Figure B.2.

These sections all met the requirements and thus provided a proper fit to the data. This fit is shown in Figure B.3.

However, because fewer sections were preferred, an attempt was made to reduce the number of sections. Because of the inaccuracy in the Mars-GRAM model close to the planet surface, it was decided to delete the outliers near the surface (section 1) and stretch the outcome of the second polynomial fit to the surface instead. Also, it turned out that section 4 and 5 could be combined in such a way that the requirements were still met. This reduced the number of section from 7 to 5. The final fit was presented in Section 3.4.1 as well as the corresponding errors and polynomial coefficients.

B.2. DENSITY

For the density curve, because the data represents a logarithmic curve, initially an exponential atmosphere was fit. However, this did not meet any of the requirements for a proper fit as can be seen in . This is why a polynomial fit was attempted for the density curve as well with the same requirement as the temperature fits. The first fit was attempted with two sections as presented in Figure B.4.

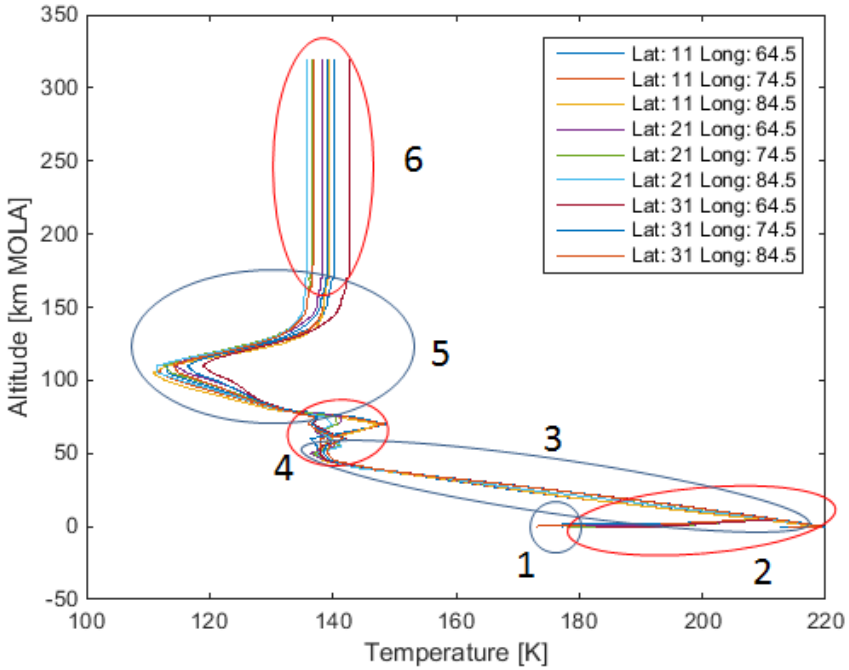


Figure B.1: Six different temperature curve sections

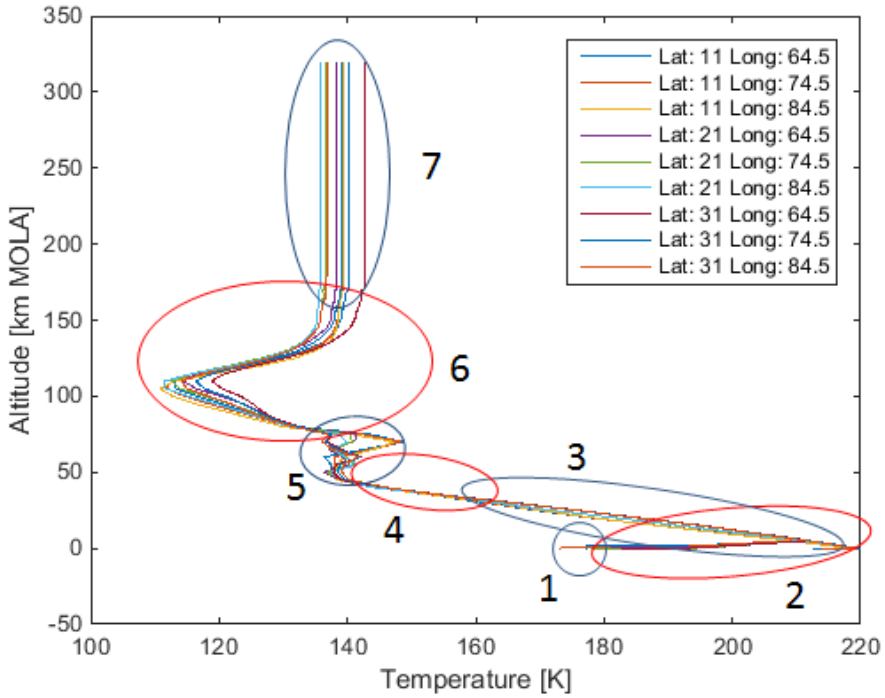


Figure B.2: Seven different temperature curve sections

However, the second section could not meet the requirements because of the initial part. This is why it was split into two sections as presented in Figure B.5.

This did meet the error requirements and resulted in the fit as shown in Figure B.6 with section 1 being

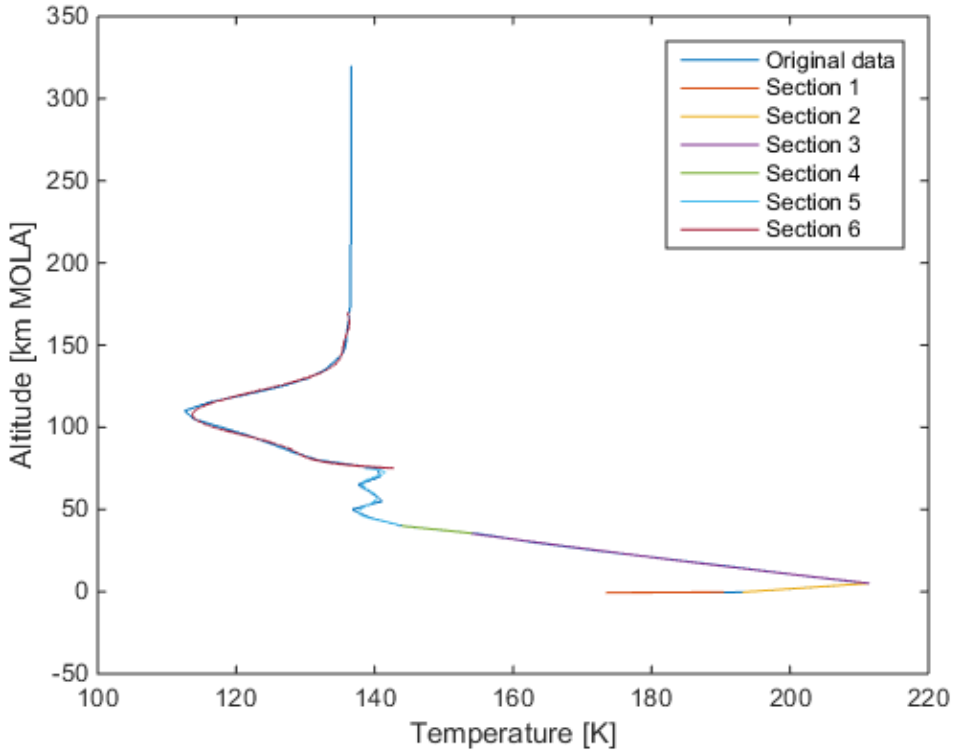


Figure B.3: All section fits for the launch site temperature data curve

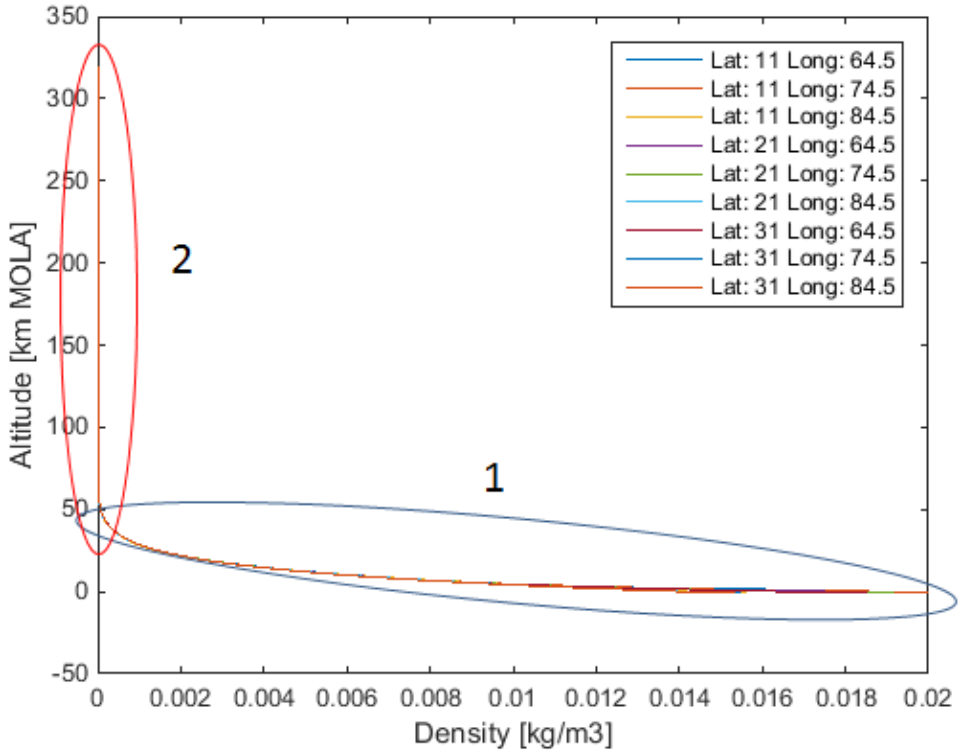


Figure B.4: Two different density curve sections

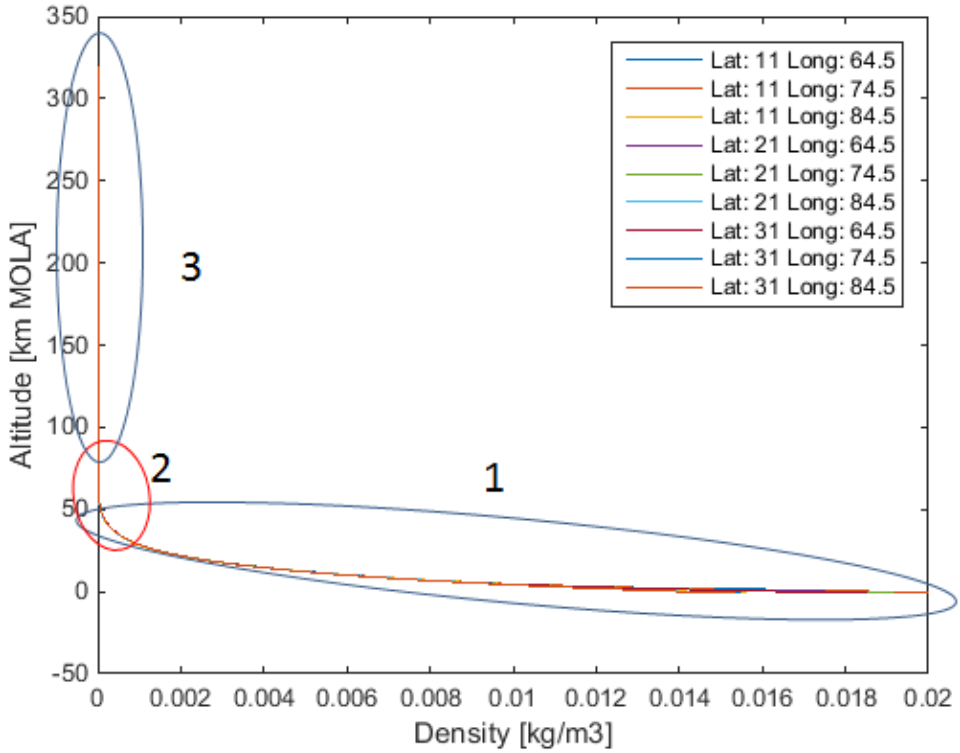


Figure B.5: Three different density curve sections

the lower part, section 2 the middle and section 3 the upper part.

However, in this case the fit for section three resulted in an oscillating behaviour around the actual data which caused the density values to drop below zero as shown in Figure B.7. Since this is unrealistic, an exponential atmospheric fit for only the third section was attempted (also shown in Figure B.7) however, this resulted in the same behaviour as the attempt for the entire curve. Therefore, it was decided to try a different exponential approach which eventually resulted in the fit described in Section 3.4.1.

Figure B.8 clearly shows that both the full exponential atmospheric fit and the polynomial fits were not a proper choice here. The figure shows the curved part of the curve.

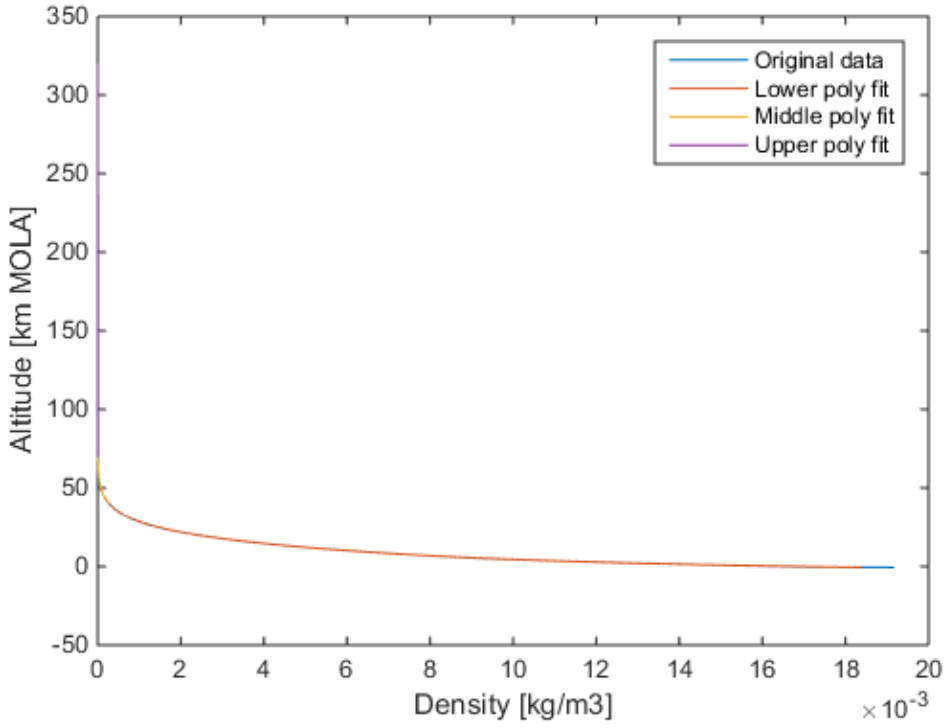


Figure B.6: All section fits for the launch site density data curve

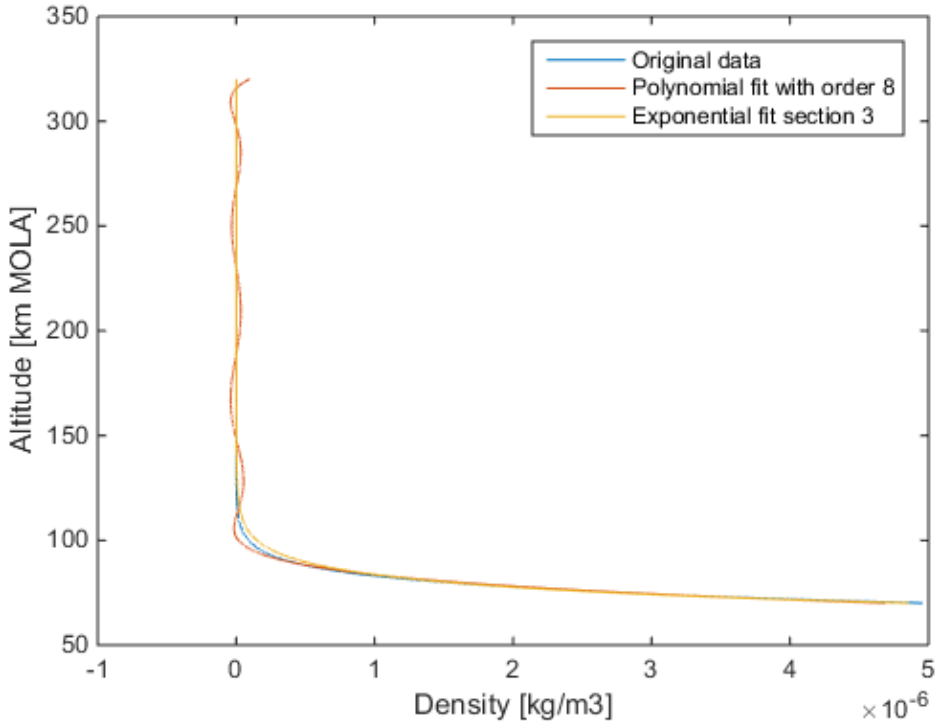


Figure B.7: Section 3 fit for the launch site density data curve

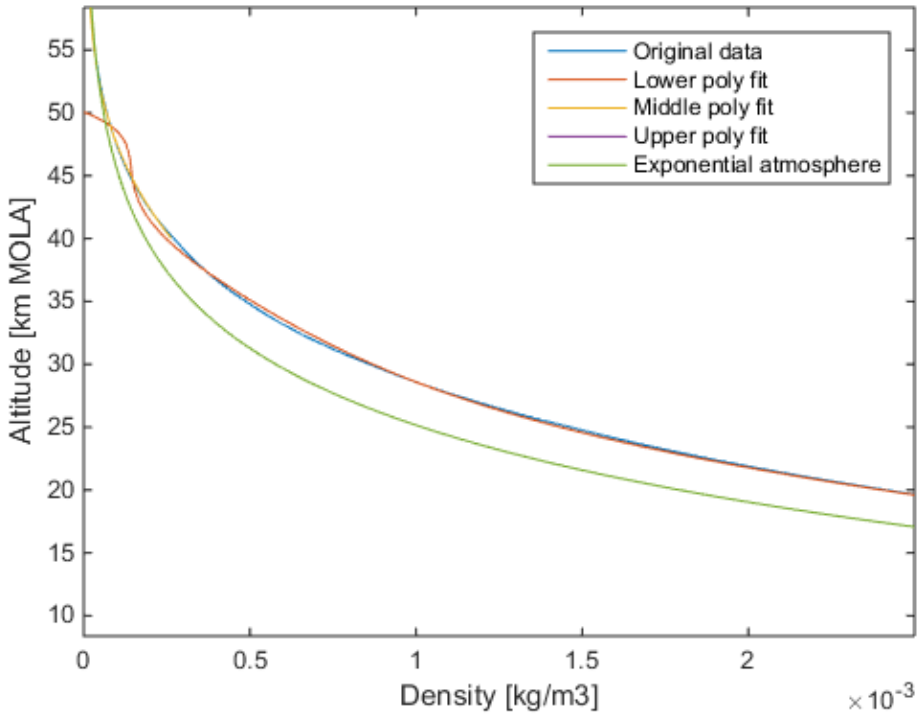


Figure B.8: Zoom in of the end of section 1 with the polynomial and exponential atmospheric fits

C

PROGRAM FILE DEFINITIONS

Both the RKF and TSI propagator architectures were presented in **??**(Figures 6.2 and 6.3 respectively). These included many different operation blocks. Each block is either a separate class, a combination of a header and source file (without it being a class), a function of either of these files or a function in the main file (MAVPropagator). The different files that include several blocks are MAVPropagator.h/.cpp, TaylorSeriesIntegration.h/.cpp, ascentDragForce.h/.cpp and ascentStateDerivativeFunction.h/.cpp, where this last file is also a class. The functions that will be implemented in these files are mentioned in Table C.1 all other blocks are described in Table C.2.

Table C.1: Large program files and included functions.

MAVPropagator	**TaylorSeriesIntegration**	**ascentStateDerivativeFunction**
Update current state	Compute auxiliaries	Compute current spherical state
Integration step	Thrust acceleration in B-frame	Compute gravitational acceleration
	Compute Taylor coefficients	Compute thrust and drag acceleration
ascentDragForce	Store state Taylor coefficients	Transfer to inertial frame
Compute speed of sound and Mach number	Initial step-size	Compute total acceleration
	Provide Taylor series expansion	Compute mass flow rate
	Estimate the max. trunc. error	
Compute drag force	Taylor series expansion incl. trunc. error	

Table C.2: Separate function files

Block	**Kind of file**
Planet characteristics	Class
Vehicle characteristics	Class
Auxiliary equations, derivatives and functions	Class
Current state and time	Class
Basic recurrence relations	Header and source
Full set of recurrence relations	Header and source
Determine next step-size	Header and source
Compute local air temperature	Header and source
Compute local air density	Header and source
Compute drag coefficient	Header and source

D

ALL RECURRENCE RELATIONS

In this appendix, all the reduced auxiliary functions that form the complete recurrence relations for each of the variables using the basic recurrence relations are provided for all three cases. Each case contains a table with the function definition, the corresponding equation and the basic recurrence relation category. All the reduced auxiliary derivatives are also described. All these expressions are a function of k, where $W(k)$ refers to the k^{th} order reduced derivative. The same holds for $X(k)$ and $U(k)$. All the expressions hold for $k \geq 1$. In the tables, the (k) is neglected to make it more readable.

D.1. FIRST CARTESIAN CASE: EULER ANGLES

The first Cartesian case requires a large number of reduced auxiliary functions. These are all provided in Table D.1. For each function, the number, the equations and the category is provided.

Table D.1: Reduced auxiliary functions for the first Cartesian case.

Auxiliary function	**Equation**	**Category**
$W_{4,1} =$	$X_1^2 + X_2^2$	Multiplication
$W_{4,2} =$	$W_{4,1} + X_3^2$	Multiplication
$W_{4,3} =$	$\sqrt{W_{4,2}}$	Power
$W_{4,4} =$	$\sqrt{W_{4,1}}$	Power
$W_{4,5} =$	$\frac{X_2}{W_{4,4}}$	Division
$W_{4,6} =$	$\frac{X_1}{W_{4,4}}$	Division
$W_{4,7} =$	$\frac{X_3}{W_{4,3}}$	Division
$W_{4,8} =$	$\frac{W_{4,4}}{W_{4,3}}$	Division
$W_{4,9} =$	$X_4 + \Omega_M X_2$	Constant Multiplication
$W_{4,10} =$	$X_5 - \Omega_M X_1$	Constant Multiplication
$W_{4,11} =$	$W_{4,9}^2 + W_{4,10}^2 + X_6^2$	Multiplication
$W_{4,12} =$	$\sqrt{W_{4,11}}$	Power
$W_{4,13} =$	$-W_{4,6} W_{4,7}$	Multiplication
$W_{4,14} =$	$-W_{4,7} W_{4,5}$	Multiplication
$W_{4,15} =$	$-W_{4,8} W_{4,6}$	Multiplication
$W_{4,16} =$	$W_{4,8} W_{4,5}$	Multiplication
$W_{4,17} =$	$X_6 W_{4,8} + W_{4,9} W_{4,13} + W_{4,10} W_{4,14}$	Multiplication
$W_{4,18} =$	$W_{4,10} W_{4,6} - W_{4,9} W_{4,5}$	Multiplication

$W_{4,19} =$	$W_{4,9}W_{4,15} - X_6 W_{4,7} + W_{4,10}W_{4,16}$	Multiplication
$W_{4,20} =$	$W_{4,17}^2 + W_{4,18}^2$	Multiplication
$W_{4,21} =$	$\sqrt{W_{4,20}}$	Power
$W_{4,22} =$	$\dfrac{W_{4,18}}{W_{4,21}}$	Division
$W_{4,23} =$	$\dfrac{W_{4,17}}{W_{4,21}}$	Division
$W_{4,24} =$	$-\dfrac{W_{4,19}}{W_{4,12}}$	Division
$W_{4,25} =$	$\dfrac{W_{4,21}}{W_{4,12}}$	Division
$W_{4,26} =$	$c\psi_T$	Cosine
$W_{4,27} =$	$c\epsilon_T$	Cosine
$W_{4,28} =$	$s\psi_T$	Sine
$W_{4,29} =$	$s\epsilon_T$	Sine
$W_{4,30} =$	$W_{4,26}W_{4,27}$	Multiplication
$W_{4,31} =$	$W_{4,27}W_{4,28}$	Multiplication
$W_{4,32} =$	$\dfrac{1}{X_7}$	Division
$W_{4,33} =$	$TW_{4,32}$	Constant Multiplication
$W_{4,34} =$	$W_{4,33}W_{4,30}$	Multiplication
$W_{4,35} =$	$\dfrac{X_{27}}{X_7}$	Division
$W_{4,36} =$	$W_{4,34} - W_{4,35}$	Subtraction
$W_{4,37} =$	$W_{4,33}W_{4,31}$	Multiplication
$W_{4,38} =$	$W_{4,33}W_{4,29}$	Multiplication
$W_{4,39} =$	$-\mu_M \dfrac{X_1}{X_9}$	Division
$W_{4,40} =$	$-W_{4,7}W_{4,23}$	Multiplication
$W_{4,41} =$	$W_{4,8}W_{4,24}$	Multiplication
$W_{4,42} =$	$-W_{4,5}W_{4,22}$	Multiplication
$W_{4,43} =$	$-W_{4,5}W_{4,23}$	Multiplication
$W_{4,44} =$	$-W_{4,8}W_{4,25}$	Multiplication
$W_{4,45} =$	$W_{4,40}W_{4,25}$	Multiplication
$W_{4,46} =$	$W_{4,42}W_{4,25}$	Multiplication
$W_{4,47} =$	$-W_{4,13}W_{4,22}$	Multiplication
$W_{4,48} =$	$W_{4,40}W_{4,24}$	Multiplication
$W_{4,49} =$	$W_{4,42}W_{4,24}$	Multiplication
$W_{4,50} =$	$W_{4,6}\left(W_{4,45} + W_{4,41}\right) + W_{4,46}$	Multiplication
$W_{4,51} =$	$W_{4,6}\left(W_{4,48} + W_{4,44}\right) + W_{4,49}$	Multiplication
$W_{4,52} =$	$W_{4,39} + W_{4,36}W_{4,50} + W_{4,37}\left(W_{4,47} + W_{4,43}\right) - W_{4,38}W_{4,51}$	Multiplication
$W_{5,1} =$	$-\mu_M \dfrac{X_2}{X_9}$	Division
$W_{5,2} =$	$W_{4,6}W_{4,22}$	Multiplication
$W_{5,3} =$	$W_{4,5}\left(W_{4,45} + W_{4,41}\right) + W_{5,2}W_{4,25}$	Multiplication
$W_{5,4} =$	$-W_{4,14}W_{4,22} + W_{4,6}W_{4,23}$	Multiplication
$W_{5,5} =$	$W_{4,5}\left(W_{4,48} + W_{4,44}\right) + W_{5,2}W_{4,24}$	Multiplication
$W_{5,6} =$	$W_{5,1} + W_{4,36}W_{5,3} + W_{4,37}W_{5,4} - W_{4,38}W_{5,5}$	Multiplication
$W_{6,1} =$	$-\mu_M \dfrac{X_3}{X_9}$	Division
$W_{6,2} =$	$W_{4,7}W_{4,24}$	Multiplication
$W_{6,3} =$	$W_{4,8}W_{4,22}$	Multiplication
$W_{6,4} =$	$-W_{4,7}W_{4,25}$	Multiplication
$W_{6,5} =$	$-W_{4,44}W_{4,23} + W_{6,2}$	Multiplication

$W_{6,6} =$	$W_{4,41} W_{4,23} + W_{6,4}$	Multiplication
$W_{6,7} =$	$W_{6,1} + W_{4,36} W_{6,5} - W_{4,37} W_{6,3} - W_{4,38} W_{6,6}$	Multiplication
$W_{8,1} =$	$X_1 X_4$	Multiplication
$W_{8,2} =$	$X_2 X_5$	Multiplication
$W_{8,3} =$	$X_3 X_6$	Multiplication
$W_{9,0} =$	$X_9 U_8$	Multiplication
$W_{9,1} =$	$\frac{W_{9,0}}{X_8}$	Division
$W_{27,1} =$	$W_{4,3}$	Redefining
$W_{27,2} =$	$W_{27,1}^2$	Multiplication
$W_{27,3} =$	$W_{27,1}^3$	Power
$W_{27,4} =$	$W_{27,1}^4$	Power
$W_{27,5} =$	$W_{27,1}^5$	Power
$W_{27,6} =$	$W_{27,1}^6$	Power
$W_{27,7} =$	$W_{27,1}^7$	Power
$W_{27,8} =$	$W_{27,1}^8$	Power
$W_{27,9} =$	$W_{27,1}^9$	Power
$W_{27,10} =$	$W_{27,1}^{10}$	Power
$W_{27,11} =$	$P_{\rho 10} W_{27,10} + P_{\rho 9} W_{27,9} + \ldots + P_{\rho 1} W_{27,1} + P_{\rho 0}$	Constant Multiplication
$W_{27,12} =$	$e^{W_{27,11}}$	Exponential
$W_{27,13} =$	$T_a = \begin{cases} P_{T1,1} W_{27,1}, & \text{for } -0.6 \le h < 5.04 \\ P_{T3,2} W_{27,3} + P_{T2,2} W_{27,2} + P_{T1,2} W_{27,1}, & \text{for } 5.04 \le h < 35.53 \\ P_{T6,3} W_{27,6} + P_{T5,3} W_{27,5} + P_{T4,3} W_{27,4} + \ldots & \\ \quad \ldots + P_{T3,3} W_{27,3} + P_{T2,3} W_{27,2} + P_{T1,3} W_{27,1}, & \text{for } 35.53 \le h < 75.07 \\ P_{T8,4} W_{27,8} + P_{T7,4} W_{27,7} + P_{T6,4} W_{27,6} + \ldots & \\ \quad \ldots + P_{T5,4} W_{27,5} + P_{T4,4} W_{27,4} + P_{T3,4} W_{27,3} + \ldots & \\ \quad \ldots + P_{T2,4} W_{27,2} + P_{T1,4} W_{27,1}, & \text{for } 75.07 \le h < 170.05 \\ 0, & \text{for } h \ge 170.05 \end{cases}$	Constant Multiplication
$W_{27,14} =$	$\sqrt{\gamma_a R_a^* W_{27,13}}$	Power
$W_{27,15} =$	$\frac{W_{4,12}}{W_{27,14}}$	Division
$W_{27,16} =$	$C_D = \begin{cases} 0, & \text{for } 0 \le M < 0.5 \\ P_{C_D 1,2} W_{27,15}, & \text{for } 0.5 \le M < 1 \\ P_{C_D 1,3} W_{27,15}, & \text{for } 1 \le M < 1.3 \\ P_{C_D 1,4} W_{27,15}, & \text{for } 1.3 \le M < 2.5 \\ P_{C_D 1,5} W_{27,15}, & \text{for } 2.5 \le M < 4 \\ 0, & \text{for } M \ge 4 \end{cases}$	Constant Multiplication
$W_{27,17} =$	$W_{4,12}^2$	Multiplication
$W_{27,18} =$	$W_{27,17} W_{27,16}$	Multiplication
$W_{27,19} =$	$\frac{1}{2} S W_{27,18} W_{27,12}$	Multiplication

The complete recurrence relation for each of the auxiliary variables can now be written as a function of the reduced auxiliary functions. These are shown by Equation (D.1) and hold for $k \ge 1$.

$$
\begin{aligned}
U_1(k) &= X_4(k) = \frac{U_4(k-1)}{k} & U_5(k) &= W_{5,6}(k) \\
U_2(k) &= X_5(k) = \frac{U_5(k-1)}{k} & U_6(k) &= W_{6,7}(k) \\
U_3(k) &= X_6(k) = \frac{U_6(k-1)}{k} & U_7(k) &= 0 \\
U_4(k) &= W_{4,52}(k) & U_8(k) &= 2W_{8,1}(k) + 2W_{8,2}(k) + 2W_{8,3}(k) \\
& & U_9(k) &= \frac{3}{2} W_{9,1}(k)
\end{aligned}
\tag{D.1}
$$

These equations are now all a function of the recurrence relations corresponding to the rest of the auxiliary functions, which are all basic recurrence relations as mentioned in Table D.1.

D.2. SECOND CARTESIAN CASE: UNIT VECTORS

The second Cartesian case is similar to the first one, however fewer reduced auxiliary functions are needed. These are all provided in Table D.2. For each function, the number, the equations and the category is provided. The numbers are not always the same as in the first case, so these should be read as completely new reduced auxiliary functions. In this case, $u4$, $u5$ and $u6$ are included in the table, because they were defined as recurrence multiplication relations.

Table D.2: Reduced auxiliary functions for the second Cartesian case.

Function	Equation	Category
$W_{4,1} =$	$X_1^2 + X_2^2$	Multiplication
$W_{4,2} =$	$W_{4,1} + X_3^2$	Multiplication
$W_{4,3} =$	$\sqrt{W_{4,2}}$	Power
$W_{4,4} =$	$\sqrt{W_{4,1}}$	Power
$W_{4,9} =$	$X_4 + \Omega_M X_2$	Constant Multiplication
$W_{4,10} =$	$X_5 - \Omega_M X_1$	Constant Multiplication
$W_{4,11} =$	$W_{4,9}^2 + W_{4,10}^2 + X_6^2$	Multiplication
$W_{4,12} =$	$\sqrt{W_{4,11}}$	Power
$W_{4,13} =$	$\frac{W_{4,9}}{W_{4,12}}$	Division
$W_{4,14} =$	$\frac{W_{4,10}}{W_{4,12}}$	Division
$W_{4,15} =$	$\frac{X_6}{W_{4,12}}$	Division
$W_{4,16} =$	$W_{4,10} X_3 - X_6 X_2$	Multiplication
$W_{4,17} =$	$X_6 X_1 - W_{4,9} X_3$	Multiplication
$W_{4,18} =$	$W_{4,9} X_2 - W_{4,10} X_1$	Multiplication
$W_{4,19} =$	$W_{4,16}^2 + W_{4,17}^2 + W_{4,18}^2$	Multiplication
$W_{4,20} =$	$\sqrt{W_{4,19}}$	Power
$W_{4,21} =$	$\frac{W_{4,16}}{W_{4,20}}$	Division
$W_{4,22} =$	$\frac{W_{4,17}}{W_{4,20}}$	Division
$W_{4,23} =$	$\frac{W_{4,18}}{W_{4,20}}$	Division
$W_{4,24} =$	$W_{4,16} W_{4,23} - W_{4,15} W_{4,22}$	Multiplication
$W_{4,25} =$	$W_{4,15} W_{4,21} - W_{4,13} W_{4,23}$	Multiplication
$W_{4,26} =$	$c\psi_T$	Cosine
$W_{4,27} =$	$c\epsilon_T$	Cosine
$W_{4,28} =$	$s\psi_T$	Sine
$W_{4,29} =$	$s\epsilon_T$	Sine
$W_{4,30} =$	$W_{4,26} W_{4,27}$	Multiplication
$W_{4,31} =$	$W_{4,27} W_{4,28}$	Multiplication
$W_{4,32} =$	$\frac{1}{X_7}$	Division
$W_{4,33} =$	$T W_{4,32}$	Constant Multiplication
$W_{4,34} =$	$W_{4,33} W_{4,30}$	Multiplication
$W_{4,35} =$	$\frac{X_{27}}{X_7}$	Division

$W_{4,36} =$	$W_{4,34} - W_{4,35}$	Subtraction
$W_{4,37} =$	$W_{4,33} W_{4,31}$	Multiplication
$W_{4,38} =$	$W_{4,33} W_{4,29}$	Multiplication
$W_{4,39} =$	$-\mu_M \frac{X_1}{X_9}$	Division
$W_{4,40} =$	$W_{4,13} W_{4,22} - W_{4,14} W_{4,21}$	Multiplication
$U_4 =$	$W_{4,39} + W_{4,36} W_{4,13} + W_{4,37} W_{4,21} - W_{4,38} W_{4,24}$	Multiplication
$W_{5,1} =$	$-\mu_M \frac{X_2}{X_9}$	Division
$U_5 =$	$W_{5,1} + W_{4,36} W_{4,14} + W_{4,37} W_{4,22} - W_{4,38} W_{4,25}$	Multiplication
$W_{6,1} =$	$-\mu_M \frac{X_3}{X_9}$	Division
$U_6 =$	$W_{6,1} + W_{4,36} W_{4,15} + W_{4,37} W_{4,23} - W_{4,38} W_{4,40}$	Multiplication
$W_{8,1} =$	$X_1 X_4$	Multiplication
$W_{8,2} =$	$X_2 X_5$	Multiplication
$W_{8,3} =$	$X_3 X_6$	Multiplication
$W_{9,0} =$	$X_9 U_8$	Multiplication
$W_{9,1} =$	$\frac{W_{9,0}}{X_8}$	Division
$W_{27,1} =$	$W_{4,3}$	Redefining
$W_{27,2} =$	$W_{27,1}^2$	Multiplication
$W_{27,3} =$	$W_{27,1}^3$	Power
$W_{27,4} =$	$W_{27,1}^4$	Power
$W_{27,5} =$	$W_{27,1}^5$	Power
$W_{27,6} =$	$W_{27,1}^6$	Power
$W_{27,7} =$	$W_{27,1}^7$	Power
$W_{27,8} =$	$W_{27,1}^8$	Power
$W_{27,9} =$	$W_{27,1}^9$	Power
$W_{27,10} =$	$W_{27,1}^{10}$	Power
$W_{27,11} =$	$P_{\rho 10} W_{27,10} + P_{\rho 9} W_{27,9} + \ldots + P_{\rho 1} W_{27,1} + P_{\rho 0}$	Constant Multiplication
$W_{27,12} =$	$e^{W_{27,11}}$	Exponential
$W_{27,13} =$	$T_a = \begin{cases} P_{T1,1} W_{27,1}, & \text{for } -0.6 \le h < 5.04 \\ P_{T3,2} W_{27,3} + P_{T2,2} W_{27,2} + P_{T1,2} W_{27,1}, & \text{for } 5.04 \le h < 35.53 \\ P_{T6,3} W_{27,6} + P_{T5,3} W_{27,5} + P_{T4,3} W_{27,4} + \ldots & \\ \quad \ldots + P_{T3,3} W_{27,3} + P_{T2,3} W_{27,2} + P_{T1,3} W_{27,1}, & \text{for } 35.53 \le h < 75.07 \\ P_{T8,4} W_{27,8} + P_{T7,4} W_{27,7} + P_{T6,4} W_{27,6} + \ldots & \\ \quad \ldots + P_{T5,4} W_{27,5} + P_{T4,4} W_{27,4} + P_{T3,4} W_{27,3} + \ldots & \\ \quad \ldots + P_{T2,4} W_{27,2} + P_{T1,4} W_{27,1}, & \text{for } 75.07 \le h < 170.05 \\ 0, & \text{for } h \ge 170.05 \end{cases}$	Constant Multiplication
$W_{27,14} =$	$\sqrt{\gamma_a R_a^* W_{27,13}}$	Power
$W_{27,15} =$	$\frac{W_{4,12}}{W_{27,14}}$	Division
$W_{27,16} =$	$C_D = \begin{cases} 0, & \text{for } 0 \le M < 0.5 \\ P_{C_D 1,2} W_{27,15}, & \text{for } 0.5 \le M < 1 \\ P_{C_D 1,3} W_{27,15}, & \text{for } 1 \le M < 1.3 \\ P_{C_D 1,4} W_{27,15}, & \text{for } 1.3 \le M < 2.5 \\ P_{C_D 1,5} W_{27,15}, & \text{for } 2.5 \le M < 4 \\ 0, & \text{for } M \ge 4 \end{cases}$	Constant Multiplication
$W_{27,17} =$	$W_{4,12}^2$	Multiplication
$W_{27,18} =$	$W_{27,17} W_{27,16}$	Multiplication

$W_{27,19} =$	$\frac{1}{2} S W_{27,18} W_{27,12}$	Multiplication

The complete recurrence relation for each of the auxiliary variables can again be written as a function of the reduced auxiliary functions. These are shown by Equation (D.2).

$$
\begin{aligned}
U_1(k) &= X_4(k) = \frac{U_4(k-1)}{k} \\
U_2(k) &= X_5(k) = \frac{U_5(k-1)}{k} \\
U_3(k) &= X_6(k) = \frac{U_6(k-1)}{k} \\
U_4(k) &= W_{4,39}(k) + W_{4,36}(k) W_{4,13}(k) + W_{4,37}(k) W_{4,21}(k) - W_{4,38}(k) W_{4,24}(k) \\
U_5(k) &= W_{5,1}(k) + W_{4,36}(k) W_{4,14}(k) + W_{4,37}(k) W_{4,22}(k) - W_{4,38}(k) W_{4,25}(k) \\
U_6(k) &= W_{6,1}(k) + W_{4,36}(k) W_{4,15}(k) + W_{4,37}(k) W_{4,23}(k) - W_{4,38}(k) W_{4,40}(k) \\
U_7(k) &= 0 \\
U_8(k) &= 2W_{8,1}(k) + 2W_{8,2}(k) + 2W_{8,3}(k) \\
U_9(k) &= \frac{3}{2} W_{9,1}(k)
\end{aligned}
\tag{D.2}
$$

D.3. SPHERICAL CASE

Since the Spherical case is very different from the Cartesian cases, it comes with its own auxiliary functions. The reduced versions are shown in Table D.3.

Table D.3: Reduced auxiliary functions for the Spherical case.

Auxiliary function	**Equation**	**Category**
$W_{4,4} =$	sX_{12}	Sine
$W_{4,5} =$	cX_{13}	Cosine
$W_{4,6} =$	cX_{12}	Cosine
$W_{4,7} =$	sX_{14}	Sine
$W_{4,8} =$	cX_{14}	Cosine
$W_{4,9} =$	sX_{13}	Sine
$W_{4,26} =$	$c\psi_T$	Cosine
$W_{4,27} =$	$c\epsilon_T$	Cosine
$W_{4,28} =$	$s\psi_T$	Sine
$W_{4,29} =$	$s\epsilon_T$	Sine
$W_{4,32} =$	$\frac{1}{X_7}$	Division
$W_{11,0} =$	$X_{15} W_{4,8}$	Multiplication
$W_{11,1} =$	$\frac{W_{11,0}}{X_{16}}$	Division
$W_{11,2} =$	$W_{11,1} W_{4,9}$	Multiplication
$W_{11,3} =$	$\frac{W_{11,2}}{W_{4,6}}$	Division
$W_{12,1} =$	$W_{11,1} W_{4,5}$	Multiplication
$W_{13,0} =$	$W_{4,28} W_{4,27}$	Multiplication
$W_{13,1} =$	$W_{4,7} W_{4,5}$	Multiplication
$W_{13,2} =$	$\Omega_M^2 X_{16} W_{4,6}$	Multiplication
$W_{13,3} =$	$W_{13,2} W_{4,4}$	Multiplication
$W_{13,4} =$	$T W_{13,0} W_{4,32}$	Multiplication
$W_{13,5} =$	$W_{13,3} W_{4,9} + W_{13,4}$	Multiplication
$W_{13,6} =$	$\frac{W_{13,5}}{X_{15}}$	Division

$W_{13,7} =$	$-2\Omega_M W_{4,6} W_{13,1} + W_{13,6}$	Multiplication
$W_{13,8} =$	$\frac{W_{13,7}}{W_{4,8}}$	Division
$W_{13,9} =$	$\left(2\Omega_M + W_{11,3}\right) W_{4,4} + W_{13,8}$	Multiplication
$W_{14,0} =$	$T W_{4,29} W_{4,32}$	Multiplication
$W_{14,1} =$	$W_{4,6} W_{4,8} + W_{13,1} W_{4,4}$	Multiplication
$W_{14,2} =$	$-\mu_M W_{4,8}$	Constant Multiplication
$W_{14,3} =$	X_{16}^2	Multiplication
$W_{14,4} =$	$\frac{W_{14,2}}{W_{14,3}}$	Division
$W_{14,5} =$	$W_{14,4} + W_{13,2} W_{14,1} + W_{14,0}$	Multiplication
$W_{14,6} =$	$\frac{W_{14,5}}{X_{15}}$	Division
$W_{14,7} =$	$2\Omega_M W_{4,6} W_{4,9} + W_{11,1} + W_{14,6}$	Multiplication
$W_{15,0} =$	$T W_{4,26} W_{4,27} - X_{27}$	Multiplication
$W_{15,1} =$	$\frac{W_{15,0}}{X_7}$	Division
$W_{15,2} =$	$-\mu_M W_{4,7}$	Constant Multiplication
$W_{15,3} =$	$\frac{W_{15,2}}{W_{14,3}}$	Division
$W_{15,4} =$	$W_{4,8} W_{4,5}$	Multiplication
$W_{15,5} =$	$W_{4,7} W_{4,6} - W_{15,4} W_{4,4}$	Multiplication
$W_{15,6} =$	$W_{13,2} W_{15,5} + W_{15,1} + W_{15,3}$	Multiplication
$W_{16,1} =$	$X_{15} W_{4,7}$	Multiplication
$W_{27,1} =$	$X_{29} U_{28}$	Multiplication
$W_{27,2} =$	$X_{28} U_{29}$	Multiplication
$W_{27,3} =$	$X_{28} X_{29}$	Multiplication
$W_{27,4} =$	$X_{15}\left(W_{27,1} + W_{27,2}\right)$	Multiplication
$W_{27,5} =$	$W_{27,3} U_{15}$	Multiplication
$W_{27,6} =$	$X_{15}\left(W_{27,4} + W_{27,5}\right)$	Multiplication
$W_{28,1} =$	$U_{30} X_{28}$	Multiplication
$W_{29,1} =$	$\begin{cases} 0, & \text{for } 0 \le M < 0.5 \\ P_{C_D 1,2} U_{32}, & \text{for } 0.5 \le M < 1 \\ P_{C_D 1,3} U_{32}, & \text{for } 1 \le M < 1.3 \\ P_{C_D 1,4} U_{32}, & \text{for } 1.3 \le M < 2.5 \\ P_{C_D 1,5} U_{32}, & \text{for } 2.5 \le M < 4 \\ 0, & \text{for } M \ge 4 \end{cases}$	Constant Multiplication
$W_{30,1} =$	X_{31}^9	Power
$W_{30,2} =$	X_{31}^8	Power
$W_{30,3} =$	X_{31}^7	Power
$W_{30,4} =$	X_{31}^6	Power
$W_{30,5} =$	X_{31}^5	Power
$W_{30,6} =$	X_{31}^4	Power
$W_{30,7} =$	X_{31}^3	Power
$W_{30,8} =$	X_{31}^2	Multiplication
$W_{30,9} =$	$U_{31}\left(10 P_{\rho 10} W_{30,1} + \cdots + 3 P_{\rho 3} W_{30,8} + 2 P_{\rho 2} X_{31} + P_{\rho 1}\right)$	Multiplication
$W_{32,1} =$	$X_{33} U_{15}$	Multiplication
$W_{32,2} =$	$X_{15} U_{33}$	Multiplication
$W_{32,3} =$	X_{33}^2	Multiplication
$W_{32,4} =$	$\frac{W_{32,1} - W_{32,2}}{W_{32,3}}$	Division

$W_{33,1} =$	$\dfrac{U_{34}}{X_{33}}$		Division
$W_{34,1} =$	$\begin{cases} P_{T1,1}U_{31}, & \text{for } -0.6 \le h < 5.04 \\ \left(3P_{T3,2}W_{30,8} + 2P_{T2,2}X_{31}\right)U_{31} + P_{T1,2}U_{31}, & \text{for } 5.04 \le h < 35.53 \\ \left(6P_{T6,3}W_{30,5} + 5P_{T5,3}W_{30,6} + 4P_{T4,3}W_{30,7} + \ldots\right. & \\ \quad \left. \ldots + 3P_{T3,3}W_{30,8} + 2P_{T2,3}X_{31}\right)U_{31} + P_{T1,3}U_{31}, & \text{for } 35.53 \le h < 75.07 \\ \left(8P_{T8,4}W_{30,3} + 7P_{T7,4}W_{30,4} + 6P_{T6,4}W_{30,5} + 5P_{T5,4}W_{30,6} + \ldots\right. & \\ \quad \left. \ldots + 4P_{T4,4}W_{30,7} + 3P_{T3,4}W_{30,8} + 2P_{T2,4}X_{31}\right)U_{31} + P_{T1,4}U_{31}, & \text{for } 75.07 \le h < 170.05 \\ 0, & \text{for } h \ge 170.05 \end{cases}$		Multiplication

In this case, a number of extra auxiliary equations were used, which result in extra recurrence relations. These are all described in Equation (D.3) and hold for $k \ge 1$.

$$
\begin{aligned}
U_7(k) &= 0 \\
U_{11}(k) &= W_{11,3}(k) \\
U_{12}(k) &= W_{12,1}(k) \\
U_{13}(k) &= W_{13,9}(k) \\
U_{14}(k) &= W_{14,7}(k) \\
U_{15}(k) &= W_{15,6}(k) \\
U_{16}(k) &= W_{16,1}(k) \\
U_{27}(k) &= \frac{1}{2} SW_{27,6}(k) \\
U_{28}(k) &= W_{28,1}(k) \\
U_{29}(k) &= W_{29,1}(k) \\
U_{30}(k) &= W_{30,9}(k) \\
U_{31}(k) &= U_{16}(k) \\
U_{32}(k) &= W_{32,4}(k) \\
U_{33}(k) &= W_{33,1}(k) \\
U_{34}(k) &= W_{34,1}(k)
\end{aligned}
\tag{D.3}
$$

E

NOMINAL CASE INPUT VALUES

In this appendix, the input files for the two different cases are given. Table E.1 describes the definition of the input parameters and the units.

Table E.1: Description of the input parameters

Parameter	**Meaning**
desiredOrbitalAltitude	This is the altitude in km MOLA that the MAV should reach, so the altitude of the desired orbit. At the same time is is also a cut-off criteria for the simulation.
desiredInclinationDeg	This is the desired inclination in degrees of the target orbit.
initialAltitude	This is the altitude at which the simulation starts, so the launch altitude for the MAV in km MOLA.
initialLatitudeDeg	This is the launch latitude on Mars in degrees at the start of the simulation.
initialLongitudeDeg	This is the launch longitude on Mars in degrees at the start of the simulation.
FlightPathAngleDeg	This is the flight-path angle in degrees at the start of the simulation. 90 degrees is straight up and 0 degrees is horizontal.
HeadingAngleDeg	This is the angle in degrees that defines the heading of the MAV and in the simulation it is defined 0 degrees when pointing North and 90 degrees when pointing East.
initialGroundVelocity	This is the ground velocity in km/s of the MAV at the start of the simulation. Because both TSI and RKF have to do too many steps if the ground velocity starts at 0 km/s, it is better to give it a small initial velocity.
massMAV	This is the GLOM in kg of the MAV.
thrust	This is the nominal thrust in Newton of the MAV engine and is a constant throughout the simulation.
specificImpulse	This is the specific impulse in seconds of the MAV engine and is a constant throughout the simulation.
initialBurnTime	This is the burn time in seconds of the first burn from the start of the simulation until the coasting phase.
constantThrustElevationAngle	This is the thrust elevation angle in degrees as defined in the propulsion frame and is a constant throughout the simulation.
constantThrustAzimuthAngle	This is the thrust azimuth angle in degrees as defined in the propulsion frame and is a constant throughout the simulation.
maxOrder	This is the order of the TSI simulation run, or K.
chosenLocalErrorTolerance	This is the error tolerance for both TSI and RKF during the actual integration and should not go below 10^{-15}.

chosenStepSize	This is the initial step-size in seconds that TSI starts with. It is the default first step-size of the step-size class that is updated every time from then on.
setEndTime	This is a cut-off criteria and can be chosen to be a certain amount of seconds at which the simulation should stop unless one of the other cut-off criteria is met first.
RKFinitiatorTime	This is the minimum time in seconds that the initiator step should be run. The initiator step is an RKF run with an error tolerance of 10^{-15} and stops when it goes beyond the initiator time. This does not mean that it is cut-off at the initiator time exactly.
rotatingPlanet	If this value is 1, the rotation of Mars is taken into account during the simulation. If it is 0 the simulation is run with a non-rotating Mars such that the inertial frame and the rotating frame are the same.
Gravity	If this value is 1, the Martian gravitational acceleration is taken into account. If it is 0, then this is neglected.
Thrust	If this value is 1, the MAV thrust acceleration is taken into account. If it is 0, then this is neglected.
Drag	If this value is 1, the drag acceleration is taken into account. If it is 0, then this is neglected.
comparison	If this value is 1, then the comparison results between TSI and RKF are computed after each integration. If it is 0, then this is not the case.

E.1. CASE 1

```
Test case from Woolley 2015 (case 10 SSTO)
desiredOrbitalAltitude          = 390.0
desiredInclinationDeg           = 45.0
initialAltitude                 = -0.6
initialLatitudeDeg              = 0.0
initialLongitudeDeg             = 74.5
FlightPathAngleDeg              = 89.0
HeadingAngleDeg                 = 90.0
initialGroundVelocity           = 0.00001
massMAV                         = 267.4
thrust                          = 3.56
specificImpulse                 = 256
initialBurnTime                 = 142.5
constantThrustElevationAngle    = -0.184
constantThrustAzimuthAngle      = -0.299
maxOrder                        = 20
chosenLocalErrorTolerance       = 1e-15
chosenStepSize                  = 0.01
setEndTime                      = 2000.0
RKFinitiatorTime                = 1.0
rotatingPlanet                  = 1
Gravity                         = 1
Thrust                          = 1
Drag                            = 1
comparison                      = 1
```

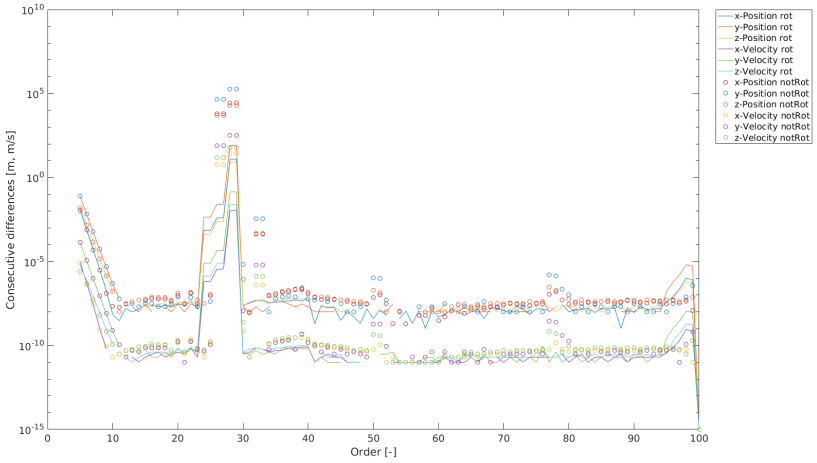
E.2. CASE 2

```
Test case from Joel (hybrid) case7_3_2016_v33
desiredOrbitalAltitude          = 479.19000000064
desiredInclinationDeg           = 92.6899999999988
initialAltitude                 = -0.6
initialLatitudeDeg              = 0.0
initialLongitudeDeg             = 90.0
FlightPathAngleDeg              = 89.0
HeadingAngleDeg                 = 90.0
initialGroundVelocity           = 0.000001
massMAV                         = 288.95985303149
thrust                          = 6.01868886452604
specificImpulse                 = 315.9
initialBurnTime                 = 99.361911852794
constantThrustElevationAngle    = -0.674
constantThrustAzimuthAngle      = 0.7273
maxOrder                        = 20
chosenLocalErrorTolerance       = 1e-15
chosenStepSize                  = 0.01
setEndTime                      = 2000.0
RKFinitiatorTime                = 1.0
rotatingPlanet                  = 1
Gravity                         = 1
Thrust                          = 1
Drag                            = 1
comparison                      = 1
```

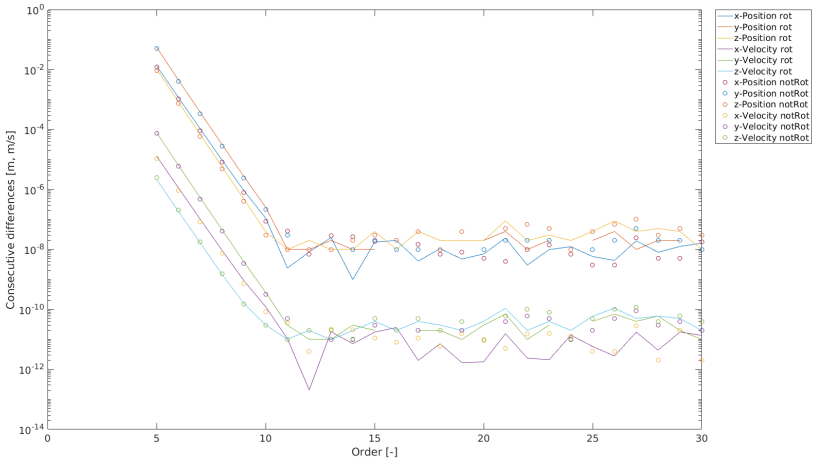
F

RESULTS

F.1. ORDER

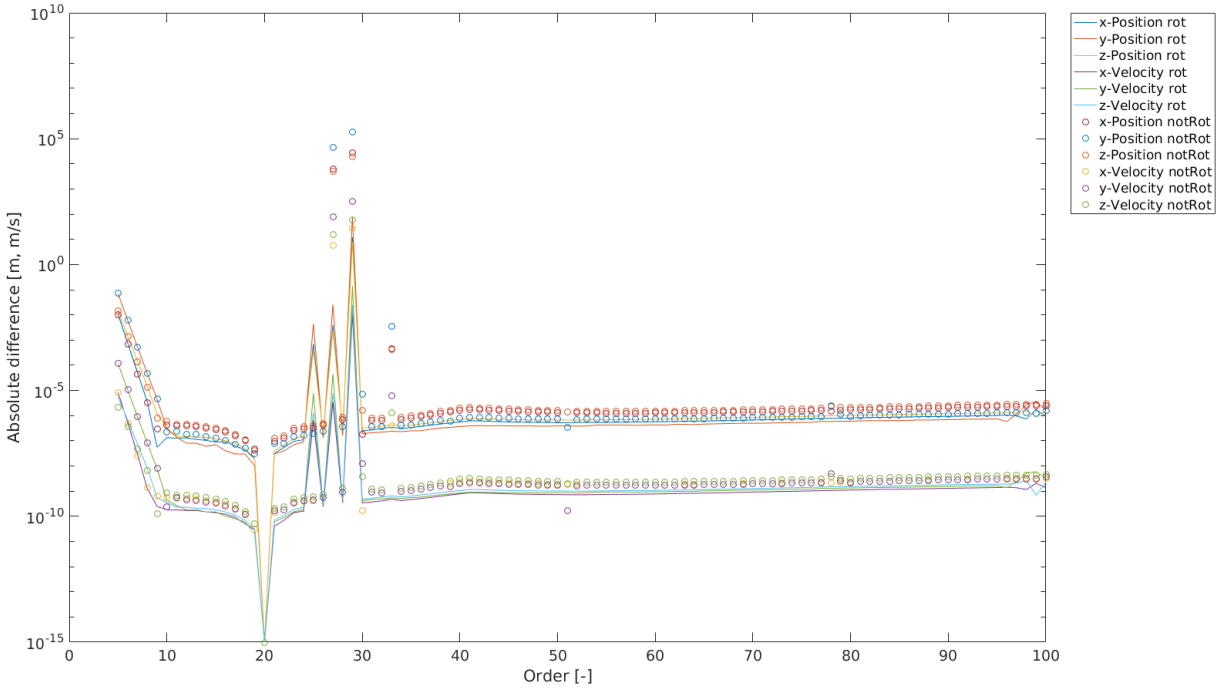


(a)

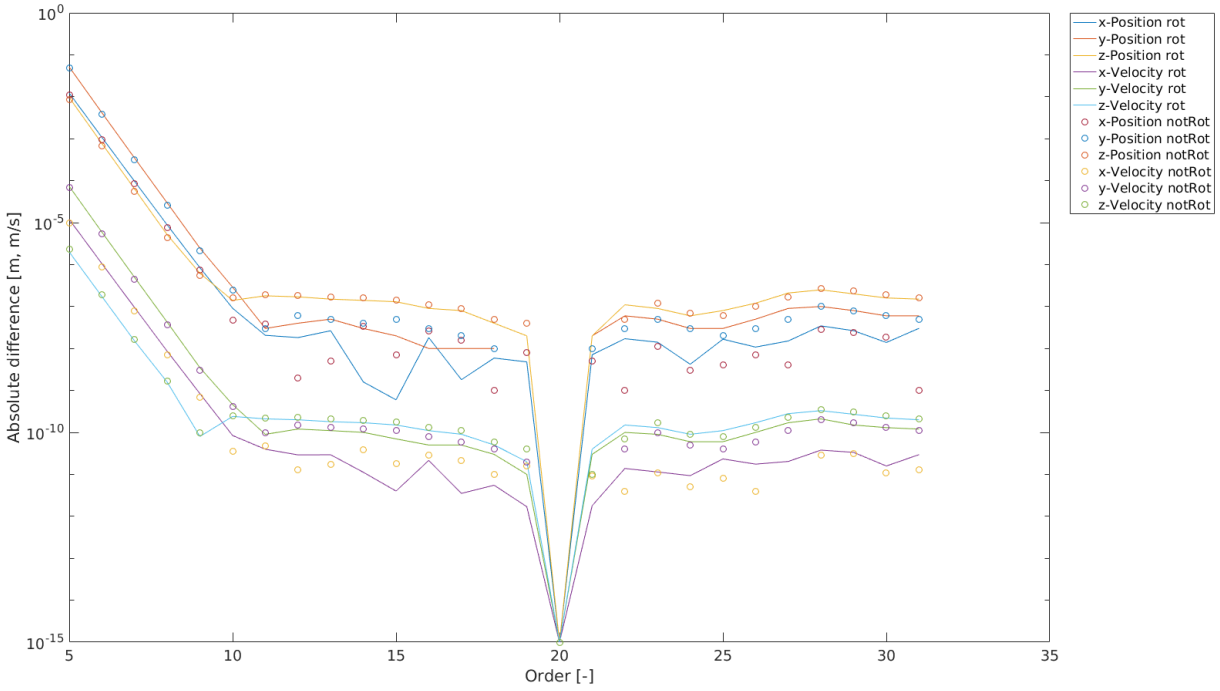


(b)

Figure F.1: Consecutive difference for rotating and non-rotating Mars (a) case 1, (b) case 2



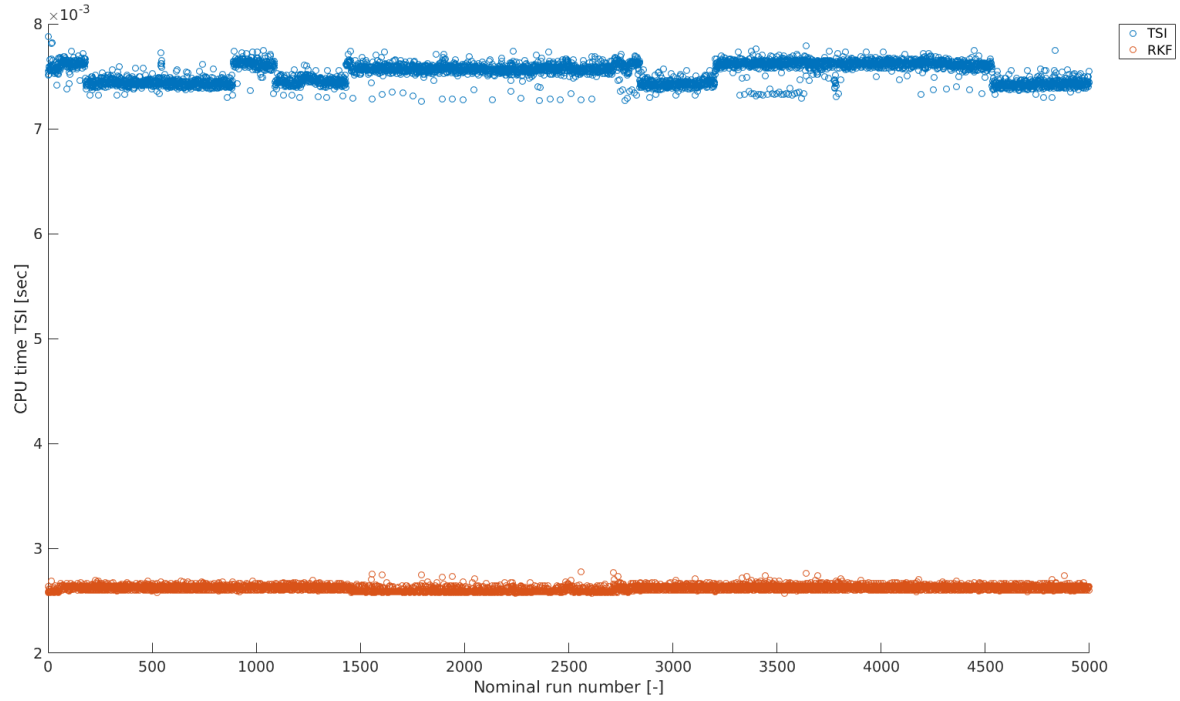
(a)



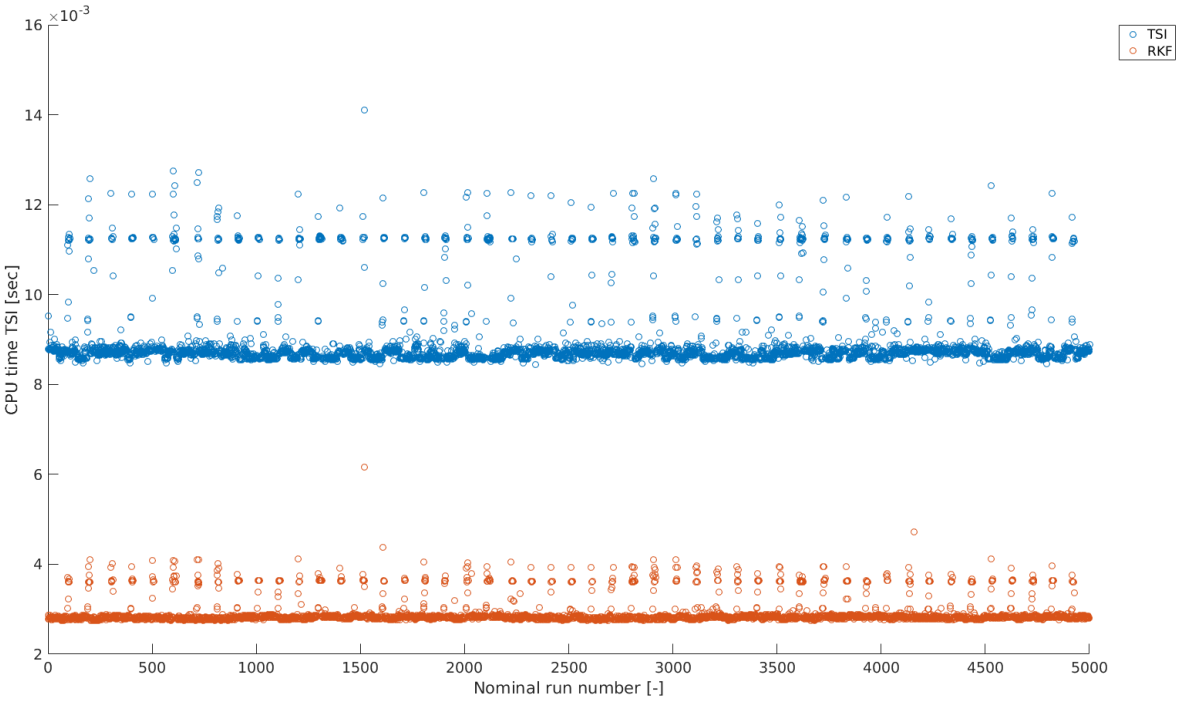
(b)

Figure F.2: Difference with respect to the nominal case for rotating and non-rotating Mars (a) case 1, (b) case 2

F.2. MULTIPLE RUNS

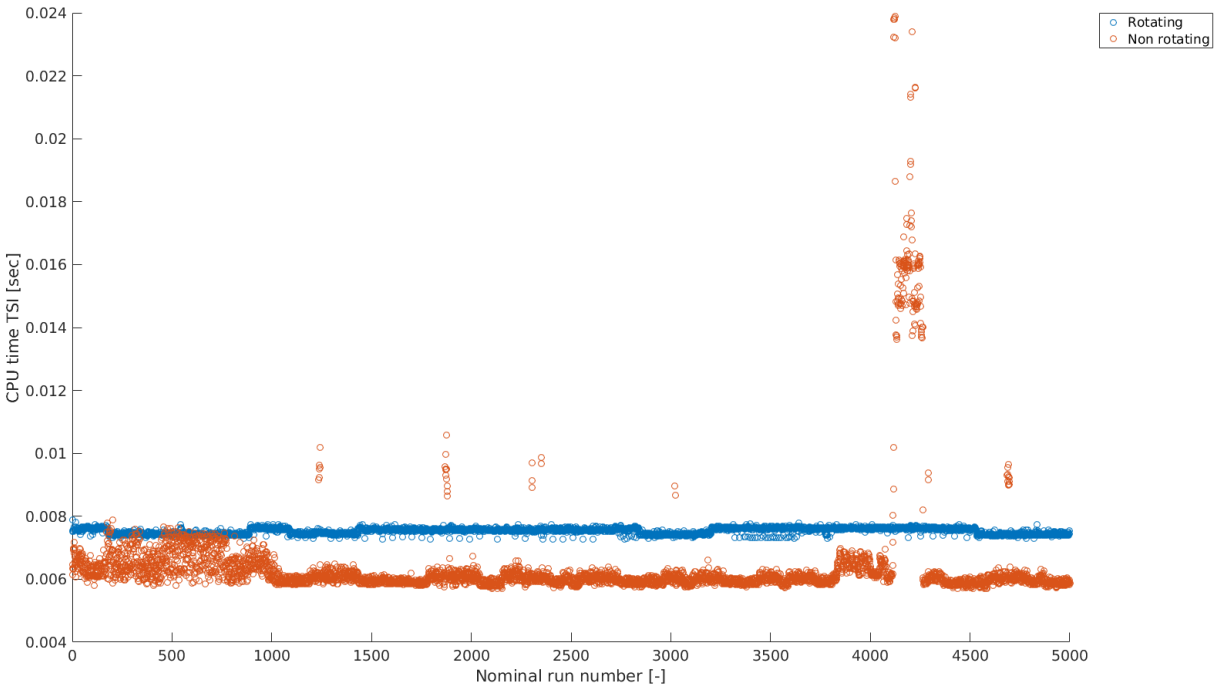


(a)

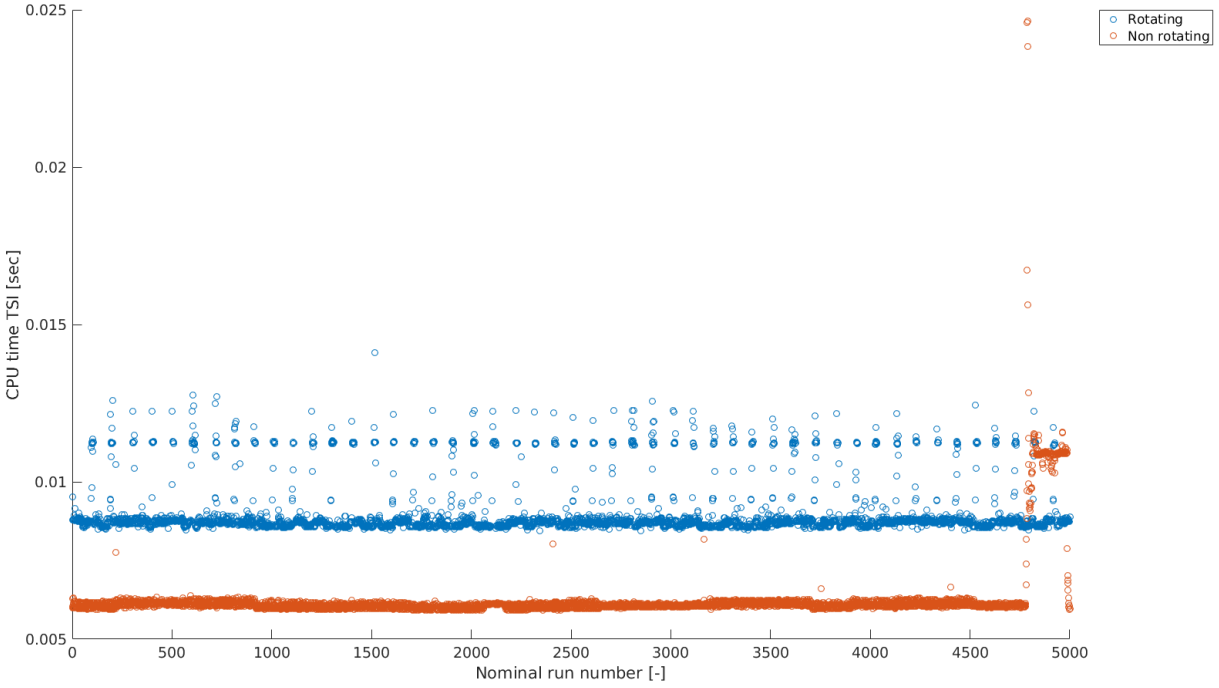


(b)

Figure F.3: Nominal runs versus CPU time comparison between RKF and TSI (a) case 1, (b) case 2



(a)



(b)

Figure F.4: Nominal runs versus CPU time for rotating and non-rotating Mars (a) case 1, (b) case 2

F.3. LAUNCH ALTITUDE

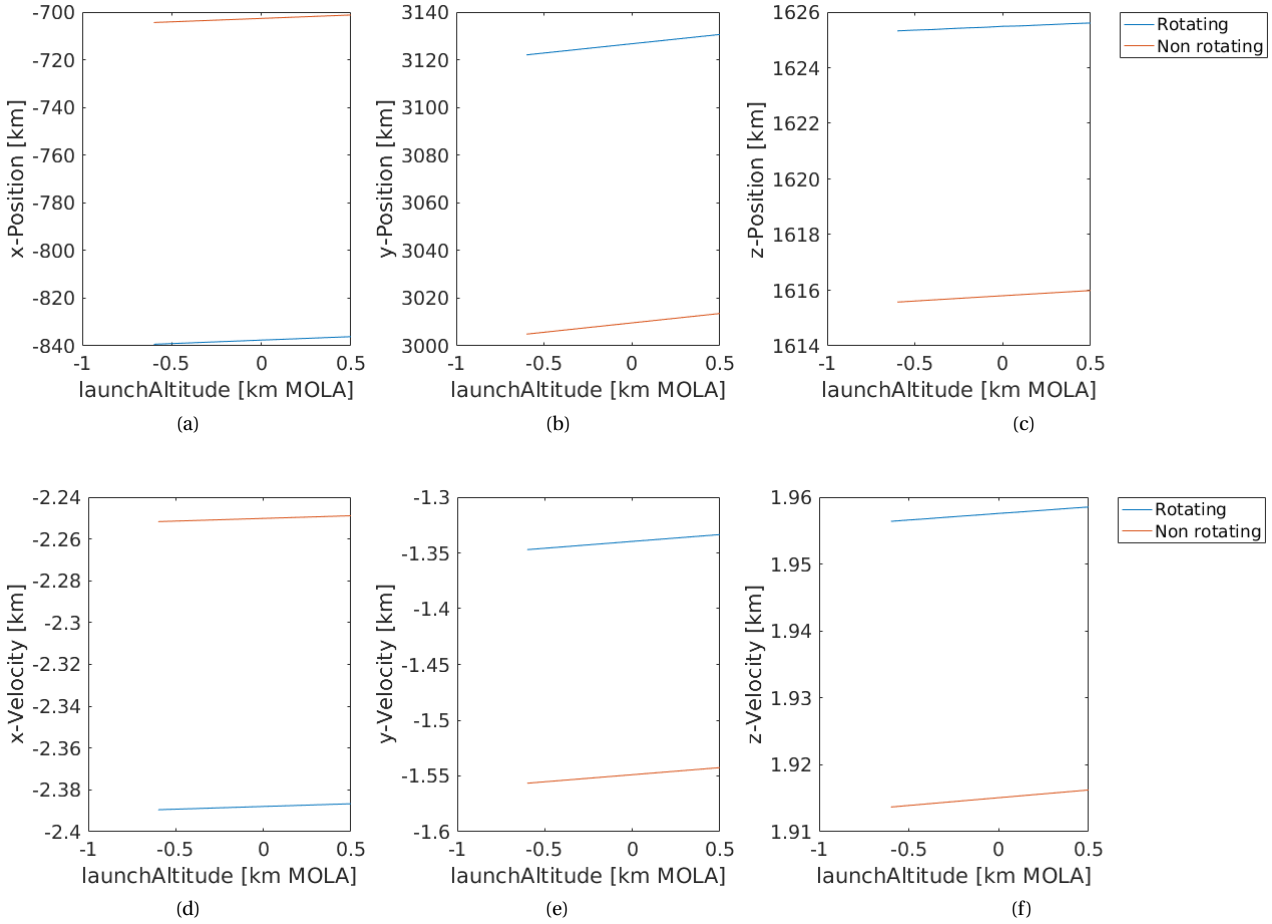


Figure F.5: Launch altitude end characteristics case 1 before circularisation for rotating and non-rotating Mars (a) x-position, (b) y-position, (c) z-position, (d) x-velocity, (e) y-velocity, (f) z-velocity

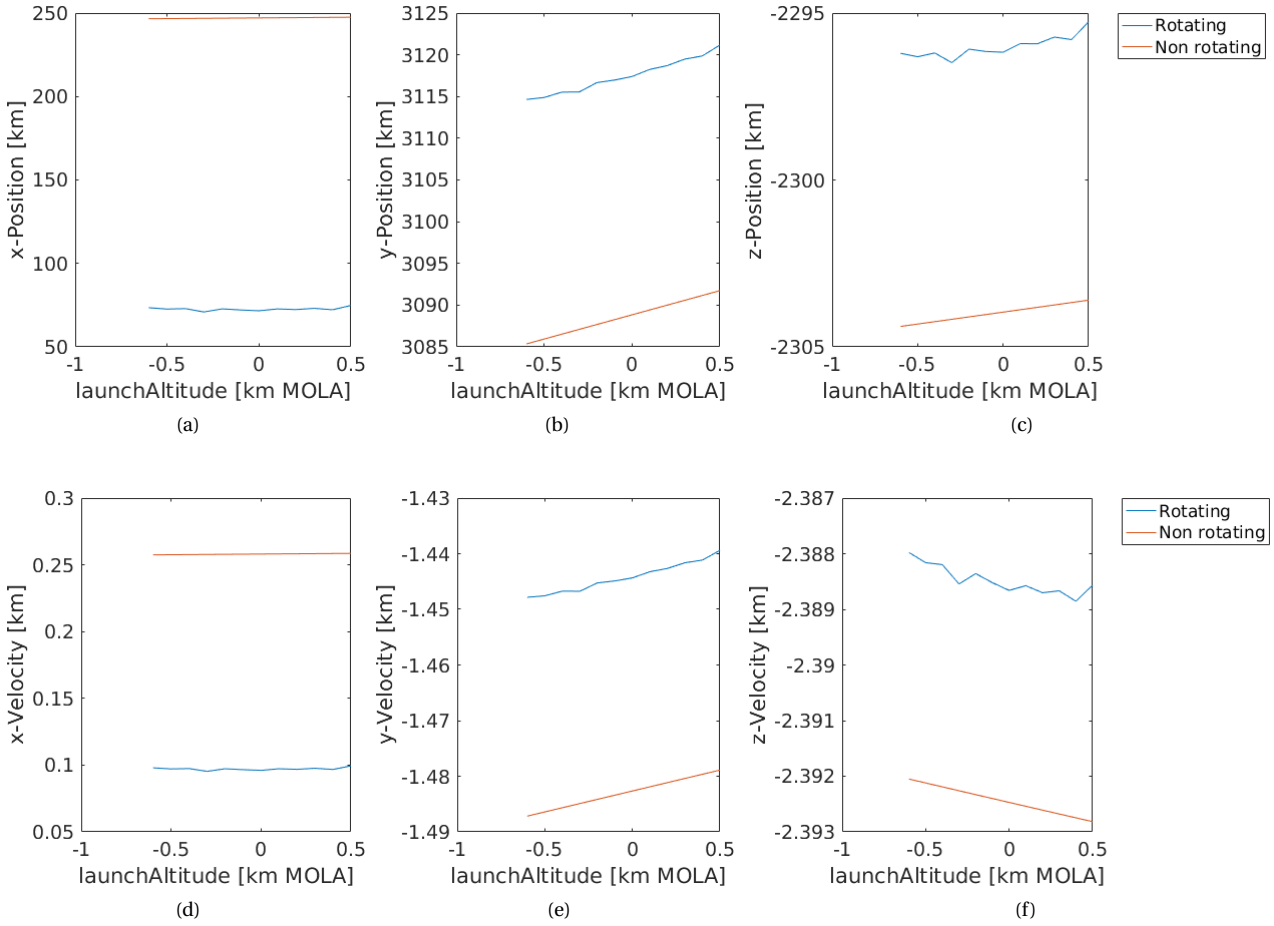


Figure F.6: Launch altitude end characteristics case 2 before circularisation for rotating and non-rotating Mars (a) x-position, (b) y-position, (c) z-position, (d) x-velocity, (e) y-velocity, (f) z-velocity

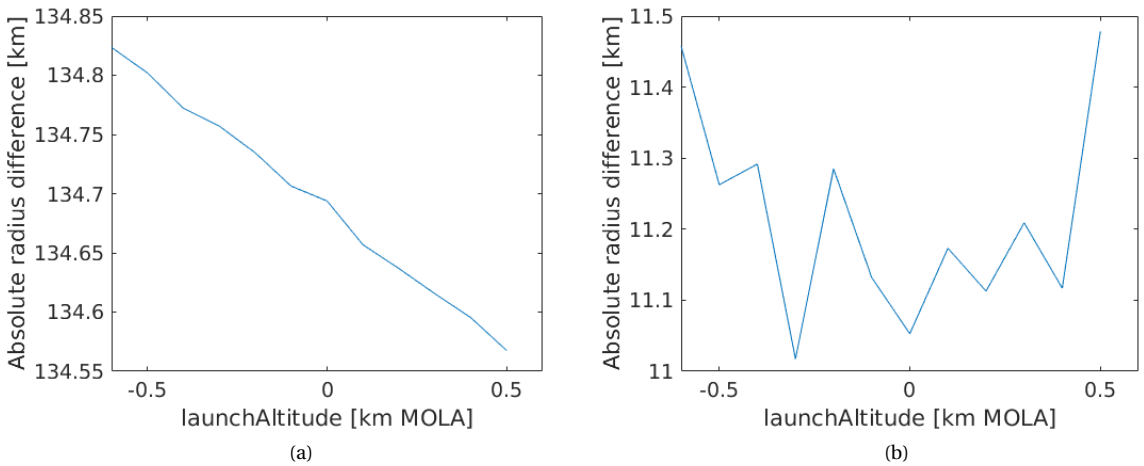


Figure F.7: Launch altitude versus radius difference between a rotating and a non-rotating Mars (a) case 1, (b) case 2

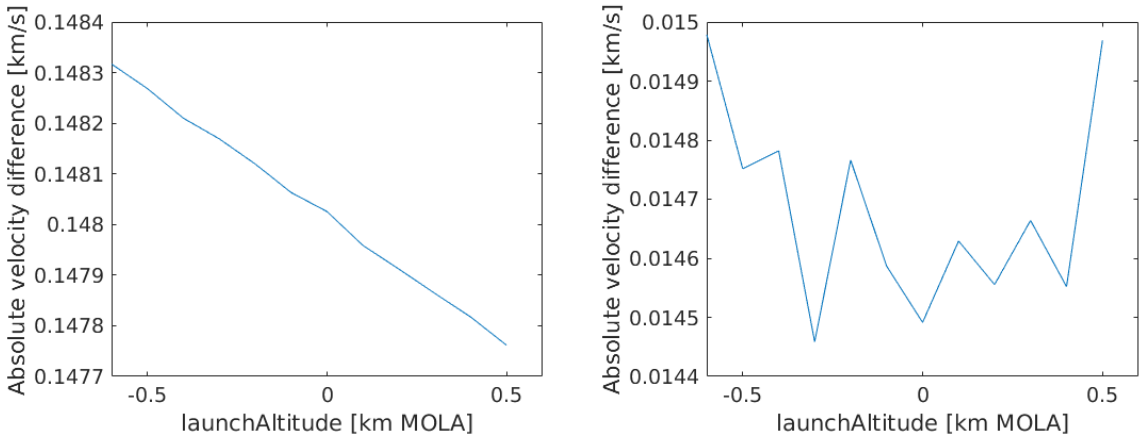


Figure F.8: Launch altitude versus velocity difference between a rotating and a non-rotating Mars (a) case 1, (b) case 2

F.4. LAUNCH LATITUDE

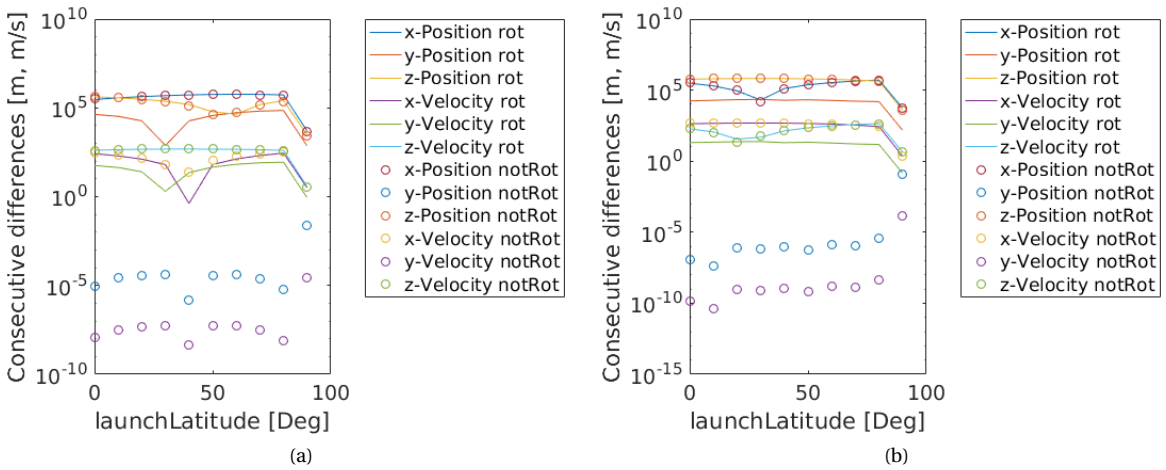


Figure F.9: Launch latitude versus consecutive difference for rotating and non-rotating Mars (a) case 1, (b) case 2

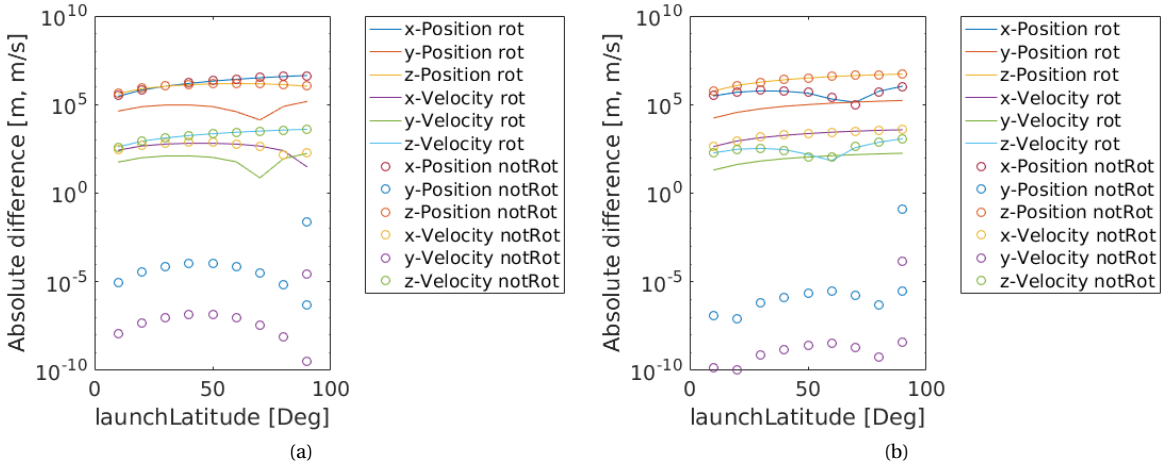


Figure F.10: Launch latitude versus nominal absolute difference for rotating and non-rotating Mars (a) case 1, (b) case 2

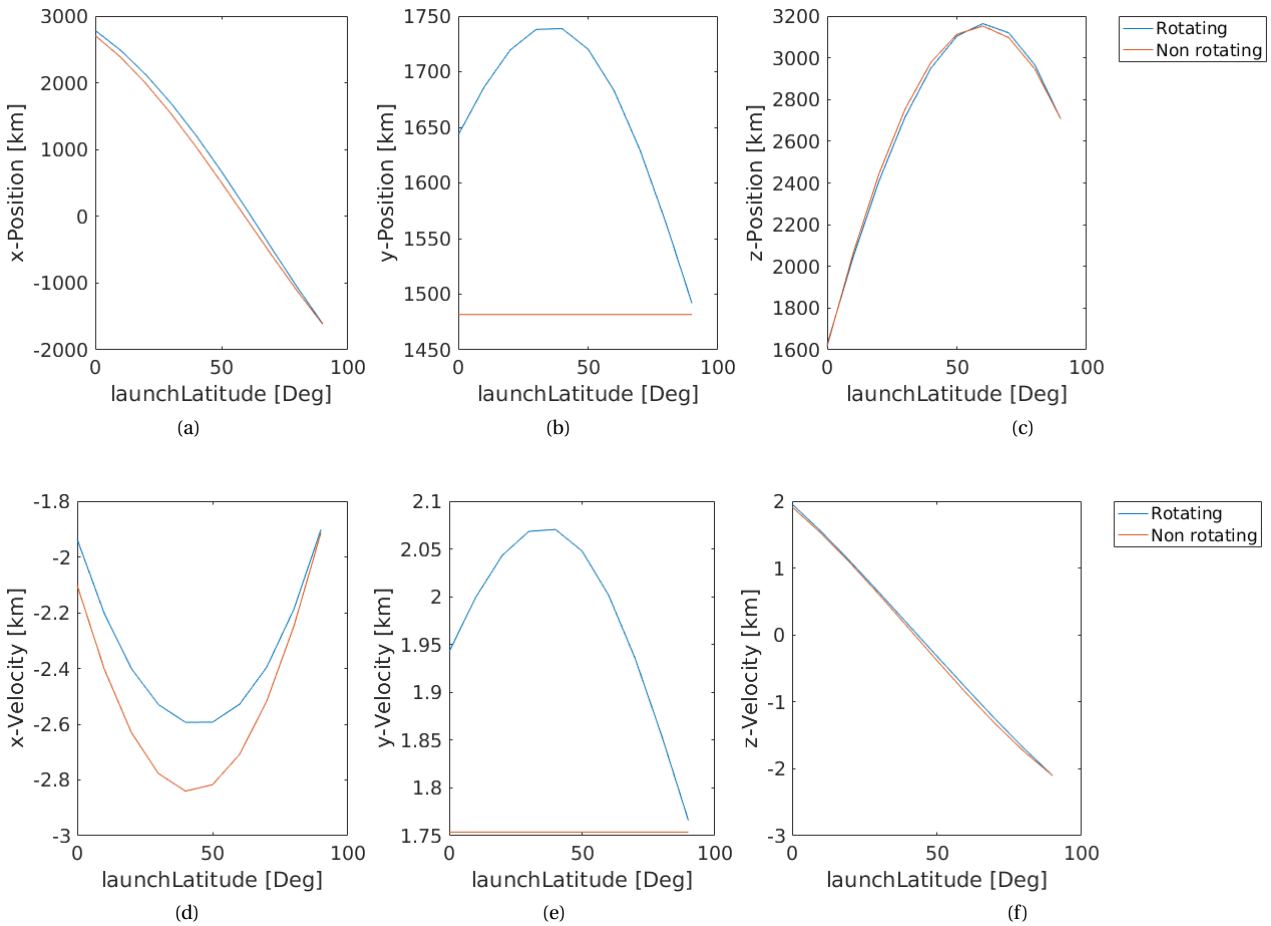


Figure F.11: Launch latitude end characteristics case 1 before circularisation for rotating and non-rotating Mars (a) x-position, (b) y-position, (c) z-position, (d) x-velocity, (e) y-velocity, (f) z-velocity

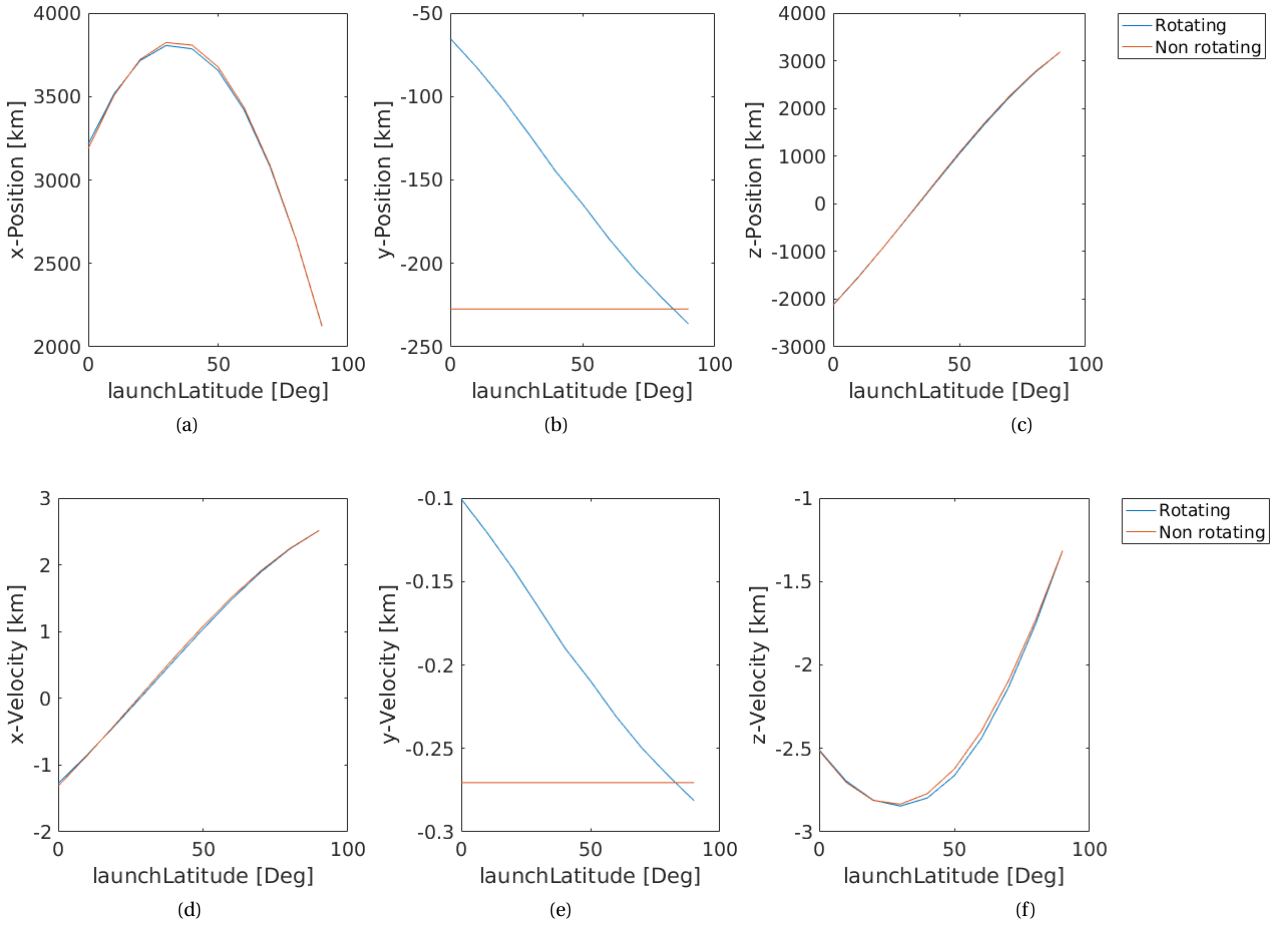


Figure F.12: Launch latitude end characteristics case 2 before circularisation for rotating and non-rotating Mars (a) x-position, (b) y-position, (c) z-position, (d) x-velocity, (e) y-velocity, (f) z-velocity

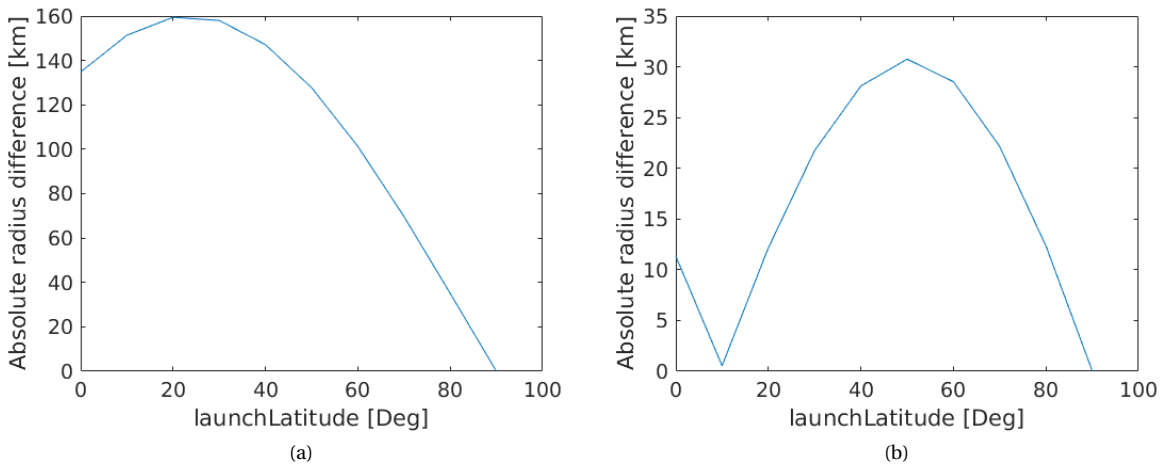


Figure F.13: Launch latitude versus radius difference between a rotating and a non-rotating Mars (a) case 1, (b) case 2

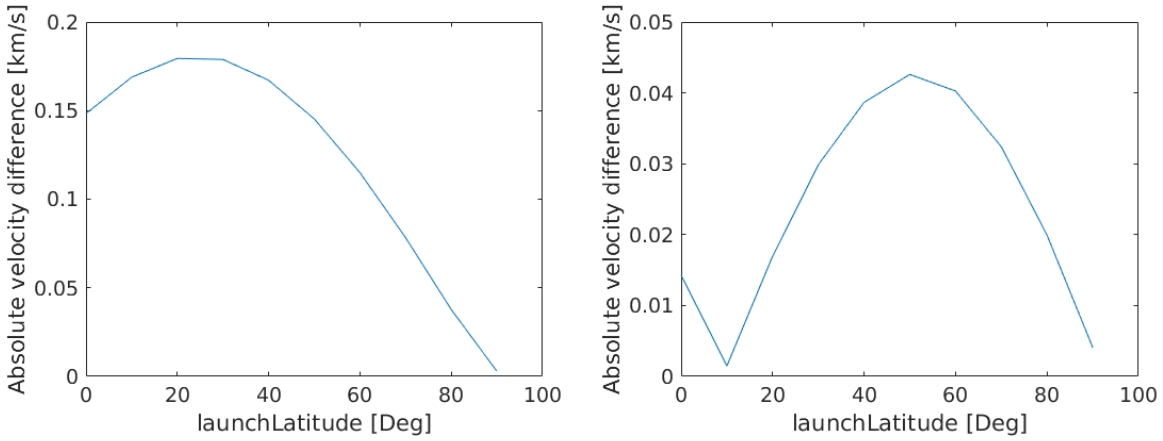


Figure F.14: Launch latitude versus velocity difference between a rotating and a non-rotating Mars (a) case 1, (b) case 2

F.5. LAUNCH LONGITUDE

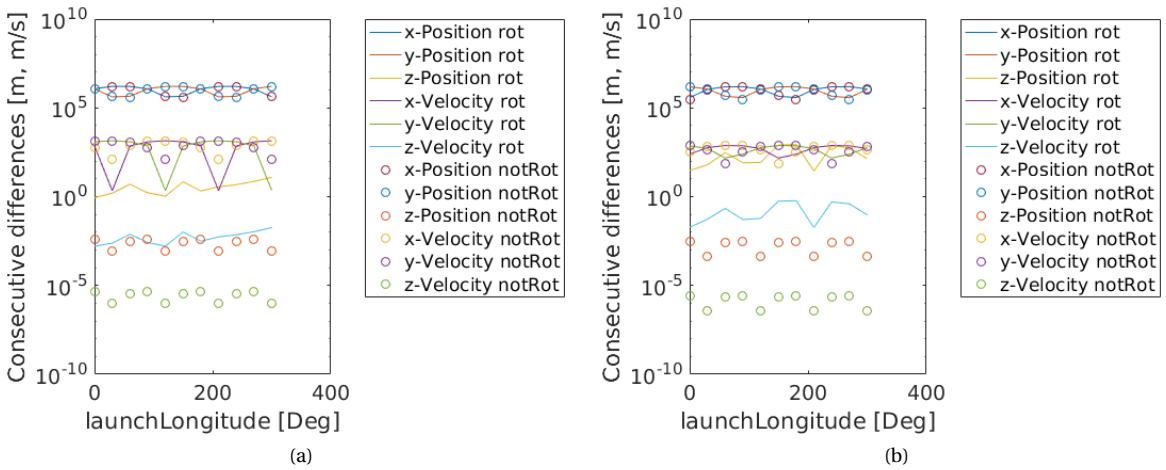


Figure F.15: Launch longitude versus consecutive difference for rotating and non-rotating Mars (a) case 1, (b) case 2

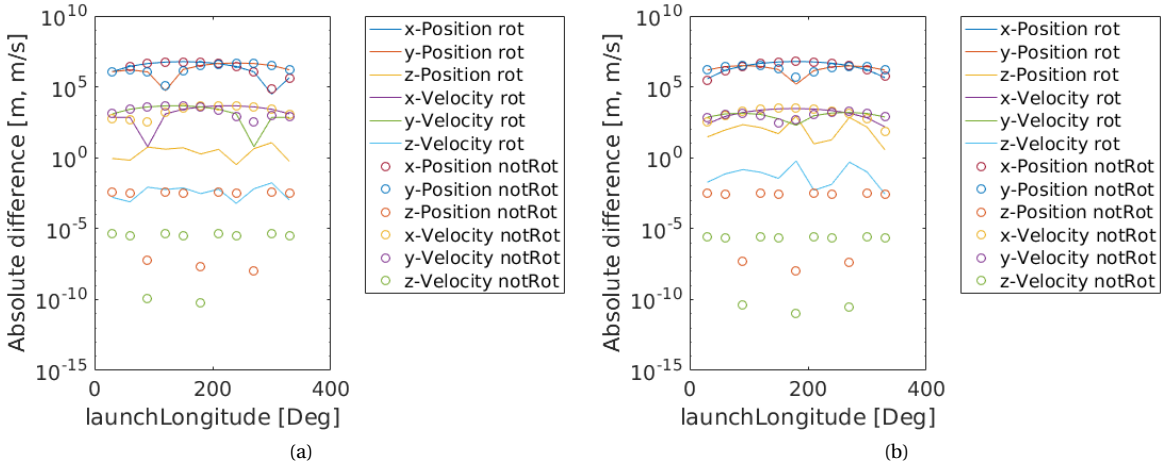


Figure F.16: Launch longitude versus nominal absolute difference for rotating and non-rotating Mars (a) case 1, (b) case 2

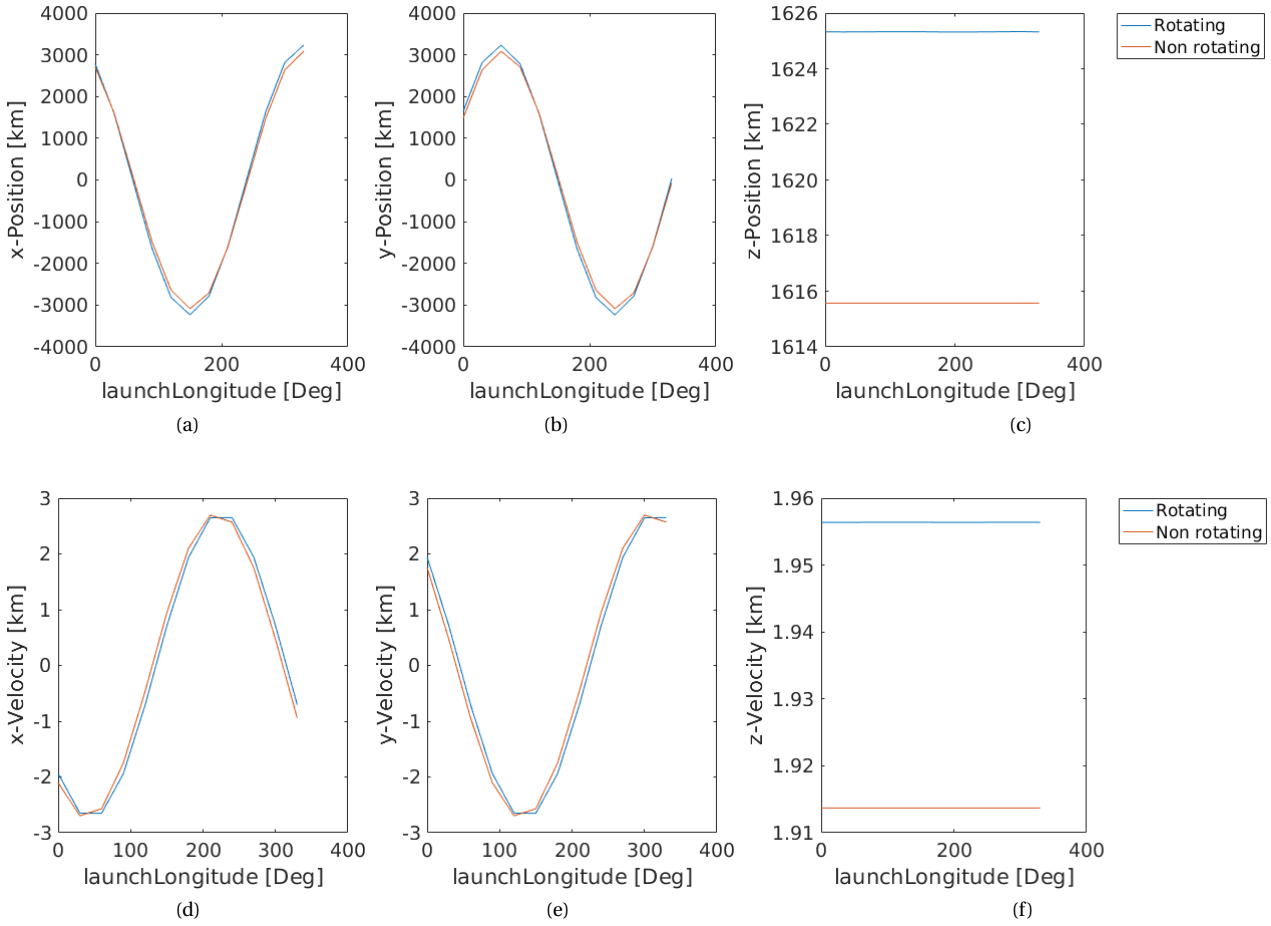


Figure F.17: Launch longitude end characteristics case 1 before circularisation for rotating and non-rotating Mars (a) x-position, (b) y-position, (c) z-position, (d) x-velocity, (e) y-velocity, (f) z-velocity

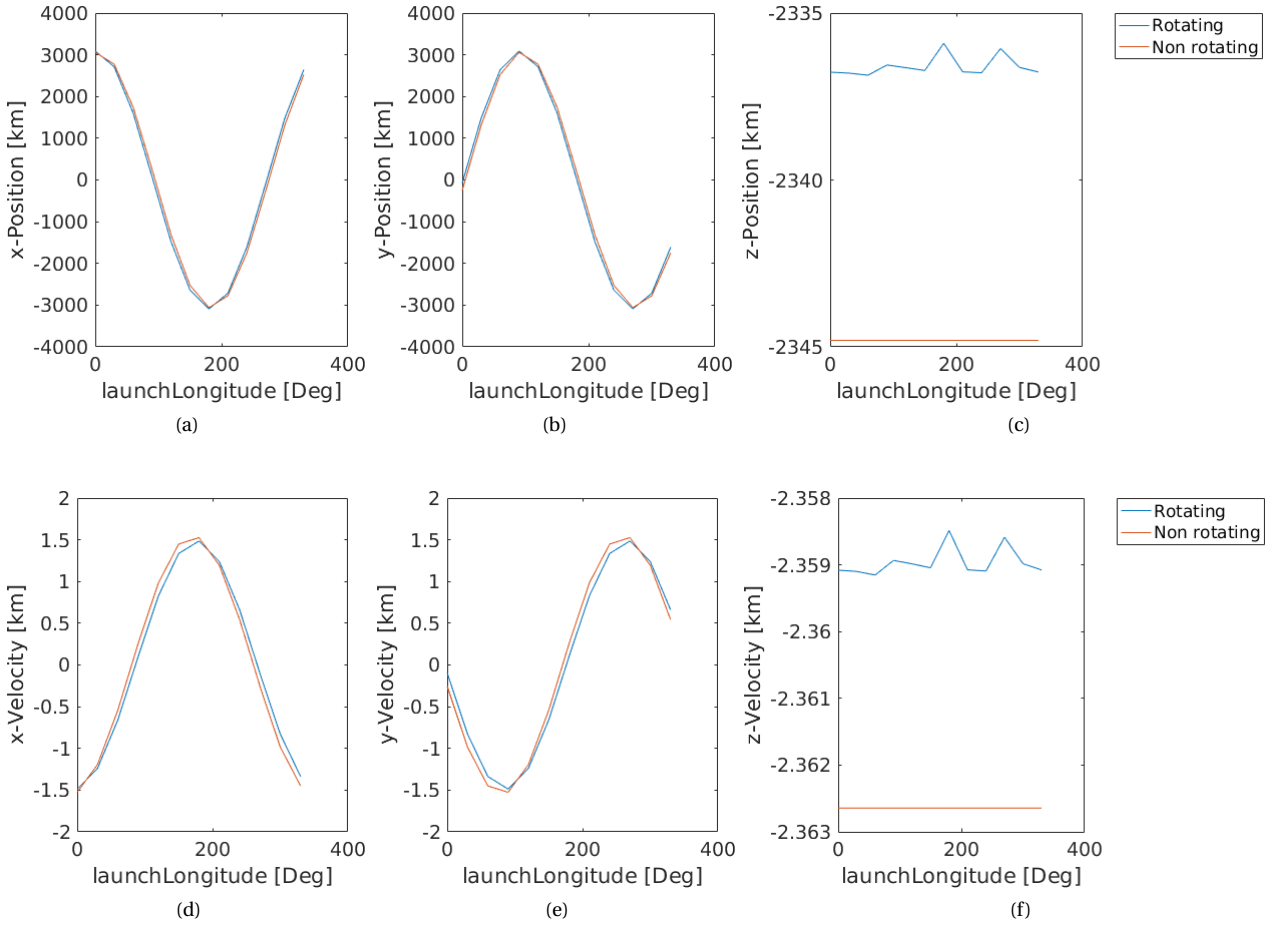


Figure F.18: Launch longitude end characteristics case 2 before circularisation for rotating and non-rotating Mars (a) x-position, (b) y-position, (c) z-position, (d) x-velocity, (e) y-velocity, (f) z-velocity

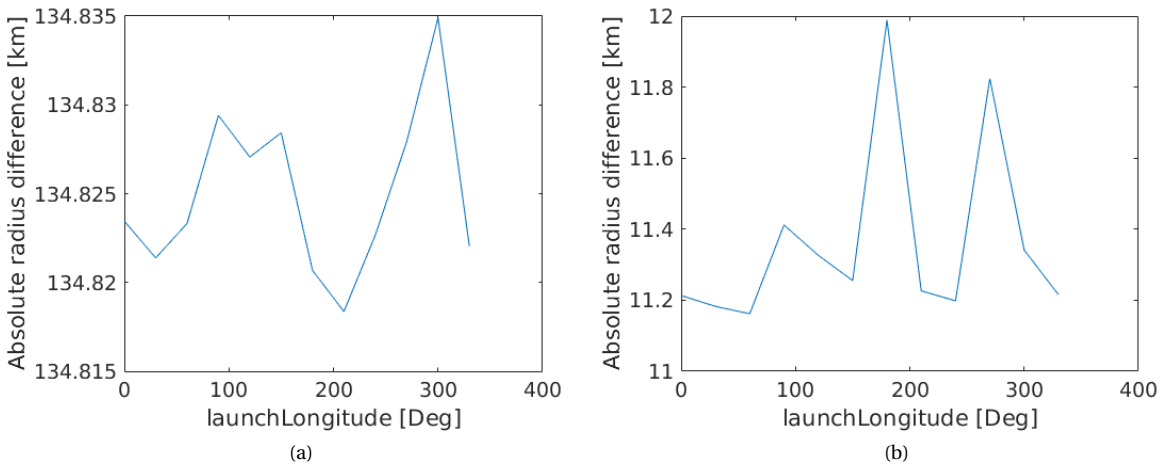


Figure F.19: Launch longitude versus radius difference between a rotating and a non-rotating Mars (a) case 1, (b) case 2

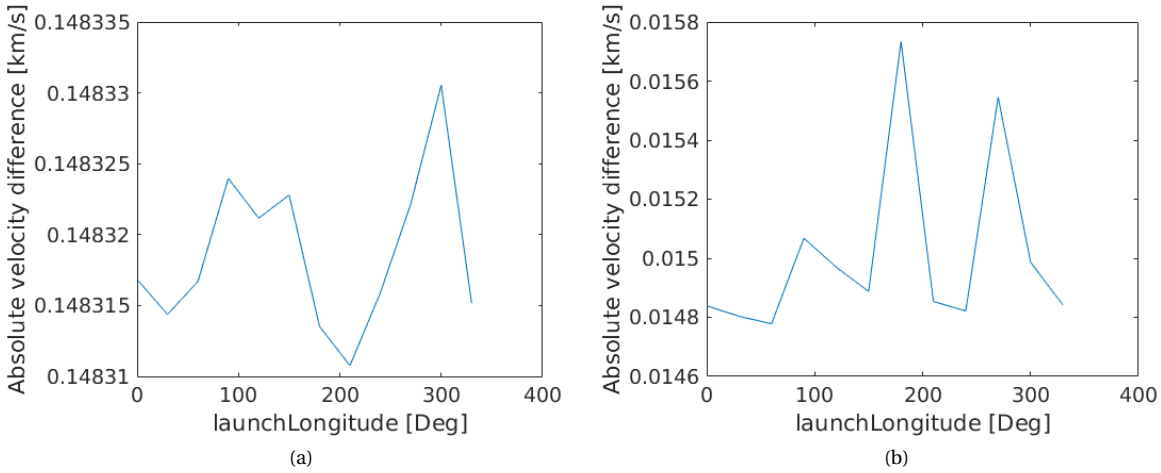


Figure F.20: Launch longitude versus velocity difference between a rotating and a non-rotating Mars (a) case 1, (b) case 2