Faster Low-Thrust Trajectory Design Through Finite Fourier Series

Thijs van Lith

Faster Low-Thrust Trajectory Design Through Finite Fourier Series

The effects of a new initialisation strategy

by

Thijs van Lith

to obtain the degree of Master of Science
at Delft University of Technology, to be defended publicly on Tuesday August 25, 2020 at 13:30.

Student number:	4359291	
Thesis committee:	Prof. dr. ir. P.N.A.M. Visser	TU Delft, Chair
	Ir. R. Noomen	TU Delft, Supervisor
	Dr. ir. R.C. Alderliesten	TU Delft

Cover image: `https://unsplash.com/photos/crs2vlkSe98`

TU Delft

Preface

This document is the result of ten months of research dedicated to analytical low-thrust transfer orbit design by means of Fourier series. It is submitted in partial fulfilment of the requirements for the degree of Master of Science at the faculty of Aerospace Engineering.

This endeavour marks the end of a six-year lasting student period in Delft. It has been a special period I will always happily and proudly cherish. I have enjoyed studying and the student life in Delft, but now it is time for the next big step. I am delighted to complete it with the study presented in this document. Even though I started studying aerospace engineering, I always have had a sincere interest in mathematics. Therefore, I am highly contented to complete my time as a student with an investigation focused on a beautiful combination of these two topics.

The final version of this thesis would of course not have been possible without the help of some special people. First and foremost I would like to express my gratitude to my admirable supervisor Ron Noomen. When I still had courses, I very much enjoyed the passion he has for space flight and the way he conveys this to his students. Now, as my supervisor, I would like to thank him for his ever stimulating and constructive attitude and ideas, especially when I thought I ran out of all my options. Furthermore, I would like to show my appreciation to Dominic Dirkx and his great assistant Elmar Puts. TUDAT is a powerful tool, specifically when it is working properly, but without the help of these two gentlemen, I would not have been able to execute my C++ code as smoothly. Then, I want to say thanks to my lovely Naomi, my parents and my brother. Even though my problems and issues were outside their field of expertise, I could still rely on a firm dose of motivation and positivity. Also, thanks to my (ex-)roommates in the most beautiful South for the great time I had. Without them the Arthur van Schendelplein would not have been so fine. Subsequently, I cannot thank my friends enough. Thanks to my aerospace buddies, who contributed to the ultimate Delft experience, thanks to my friends from CfS, who broadened my worldview with regards to the social sciences and with whom I had an unforgettable experience on the other side of the ocean, and at last, thanks to the gentlemen from the Dokter Biegelstraat in G-Town, whom I can invariably rely on.

Thijs van Lith
Delft - Thursday 23rd July, 2020

Summary

Low-thrust propulsion has gained more popularity over the past few decades because of its high efficiency. Interplanetary transfer trajectories in particular benefit from low-thrust propulsion, considering the typically high ΔV to be achieved. In order to allow a fast design of such missions, first-order, efficient representations of transfer orbits are usually used before a more detailed and exact numerical model is applied. One of these so-called shape-based methods is the finite Fourier series method is.

The finite Fourier series allows for the modelling of such trajectories by means of a set of Fourier series. These coefficients will describe the trajectory, determine the thrust profile, and most importantly: the required ΔV. Besides satisfying constraints on the initial and final position and velocity, and on the time of flight, it can also comply with optionally imposed thrust acceleration constraints. This greatly contributes to an efficient generation of realistic and feasible transfer trajectories.

In this thesis, the focus lies on improving this first-order method through a different initialisation strategy with the goal of decreasing the convergence time of the algorithm, and improving its three-dimensional stability.

The first step in this process is the implementation of the method. Both the two- and three-dimensional variant have been added to TUDAT (Technical University of Delft Astrodynamics Toolbox) and a step-by-step guide on the two versions can be read in this report.

During this process, some inconsistencies have been encountered that were not clearly, or even incorrectly addressed or documented by the inventors of this method. These include the calculation of the initial guess, the definition of the decision vector, the performance of the two-dimensional unconstrained finite Fourier series, the two-dimensional reference results and the interpretation of the reference frame.

After these problems have been overcome, the method has been validated against three case studies departing from the Earth: a trajectory to Mars, both in two and three dimensions, and an additional trajectory to the comet Tempel-1 in three dimensions. These targets were chosen to test the performance to a target with both a simple orbital geometry, i.e. low inclination and a nearly circular orbit, and targets that are more challenging at higher inclinations and with higher eccentricities.

The second step is the actual analysis of the initialisation strategy. The original strategy is based on the approximation of the trajectory by means of a third-order power function. In the search of a better strategy, four different function types have been analysed: (the original) power function, an exponential function, a trigonometric function and a logarithmic function. The results obtained from the power function obviously serve as reference data. The four different functions were tested on two transfer trajectories, both in two and three dimensions: from Earth to Jupiter and from Earth to Dionysus.

In two dimensions, the radial component r and the transfer angle θ are both approximated by the same function. This holds for the three-dimensional version as well, but it should be noted that the parameters for the axial component z are not initialised by any function and are just set to zero before they are fed to the solver. To analyse the effects of different strategies, five test cases have been defined: two in two dimensions and three in three dimensions.

The first two-dimensional case consists of using one of the four initialisation functions for both parameters, while the second case prescribes that the two parameters are initialised by two functions that are tailored to their natural behaviour. Because the radial component of a low-thrust trajectory naturally has the shape of a power function and the transfer angle like a logarithmic function, these two function types are used to initialise r and θ.

In three dimensions more or less the same protocol is followed. The first case comprises of using the same function for both r and θ while z is still set to zero. The second case also allows z to be approximated by the same initialisation function as r and θ, hence it is no longer set to zero. The third case again makes use of the natural behaviour of the three design parameters. This time it means that r is estimated by a power function, θ by a logarithmic function and z by a sinusoidal function.

The results in two dimensions look very promising, as for the first case the solver manages to find the solution of 18.30 km/s with an average of 18.22 km/s and a spread of only 0.085 km/s for the trajectory to Jupiter, and an average of 22.46 km/s with a spread of 0 km/s to Dionysus. The exponential initialisation function provides the fastest results, which is 7.9% and 42.6% faster than the respective reference case.

The second case with a tailored initialisation function, on the contrary, fails to converge when finding a trajectory to Dionysus. The best solution for a trajectory to Jupiter is 19.0 km/s, which is more than the reference case. Also, the computation time is 8.2% slower.

In three dimensions the results show a different trend. For the first case, it is only the exponential approach that finds a feasible trajectory to Jupiter besides the reference solution, but it has a ΔV outcome of 18.59 km/s against 16.74 km/s for the reference solution. Not a single feasible trajectory solution to Dionysus is found.

For the second case, the results are of a similar form. Again no feasible trajectory solution to Dionysus could be found. To Jupiter, it is only the power and exponential initialisation function that manage to find a feasible solution of 19.29 km/s and 23.69 km/s respectively.

Finally, case number three, which makes use of tailored initialisation functions, does not contain any feasible trajectory at all. For Jupiter, the solver converges to a ΔV of 126.44 km/s when the axial coefficients are set to zero and 135.02 km/s when they are estimated as well. The ΔVs for the trajectory to Dionysus are 167.13 km/s and 170.28 km/s respectively, which are all highly unrealistic.

Based on the results it can be concluded that the use of the proper initialisation strategy can considerably boost the effectiveness of the finite Fourier series method. However, a clear contrast between the two-dimensional and three-dimensional version can be observed: the two-dimensional version of the algorithm can greatly benefit from an exponential function to generate a priori values, while there has not been a single approach that decreases the convergence time of the solver nor improves the stability for the three-dimensional version.

Therefore, it can also be concluded that the finite Fourier series method is extremely sensitive regarding a priori values for the axial candidate. All cases in which the coefficients were left untouched and thus initially set to zero yield better results than when they were estimated by one of the initialisation functions.

Nomenclature

Latin Symbols

$[A_r]$	Fourier coefficient (C_{a_n}, C_{b_n}) matrix	–
A	Amplitude of trigonometric initialisation function	–
A	Cross-sectional area	m^2
$a-g$	Inverse polynomial coefficients	–
$a-h$	Cubic polynomial coefficients	–
a, b, c	Initialisation function coefficients	–
a_{0_P}	Constant used to describe out-of-plane thrust	–
a_{θ_n}	Finite polynomial design parameter for $\theta(t)$	–
a_d	Acceleration of perturbing body d acting on spacecraft	$\mathrm{m/s^2}$
a_k	Acceleration of orbited body acting on spacecraft	$\mathrm{m/s^2}$
$a_n - f_n$	Finite Fourier series coefficient	–
a_{r_m}	Finite polynomial design parameter for $r(t)$	–
$a_z - d_z$	Inverse polynomial 3D coefficients	–
b_0	Constant used to describe out-of-plane thrust	–
C	Thrust acceleration constraint	–
c	Speed of light [26]	2.99792458×10^8 km/s
C_{a_n}, C_{b_n}	Short-handed notation for Fourier term	–
c_i	Hodographic base function coefficient	–
C_R	Coefficient of reflectivity	–
d	Dimension of Schwefel function	–
$D(\theta)$	Time equation scalar function	–
$\mathbf{e_S}$	Unit vector pointing towards Sun	–
e	Eccentricity	–
$\mathbf{f}_{SRP}$	Solar Radiation Pressure	$\mathrm{m/s^2}$
F	Thrust force	N
$f(\tau)$	Initialisation function for Fourier coefficients	–
f_θ	Transverse thrust acceleration	$\mathrm{m/s^2}$
f_h	Thrust acceleration acting along or against angular momentum vector of spacecraft	$\mathrm{m/s^2}$
F_r	Short-handed notation for Fourier term	–
f_r	Radial thrust acceleration	$\mathrm{m/s^2}$
f_z	Axial thrust acceleration	$\mathrm{m/s^2}$
g_0	Standard Earth gravity	$\mathrm{m/s^2}$
i	Inclination	rad
I_{sp}	Specific impulse	s
K	Mean of trigonometric initialisation function	–
k_0	Scaling factor	m
k_1	Dynamic range parameter	–
k_2	Winding parameter	–
$\dot{m}$	Mass flow	kg/s
m	Mass	kg
m	Number of discretisation points	–
m_d	Mass of perturbing body	kg
m_k	Mass of orbited body	kg
n	Thrust direction specifier	–
n	n^{th} Fourier coefficient	–
N_{rev}	Number of revolutions	–
n_r	Number of Fourier coefficients for $r(t)$	–
n_θ	Number of Fourier coefficients for $\theta(t)$	–

n_{app}	Number of discretised data points to initialise finite Fourier series	–
n_z	Number of Fourier coefficients for $z(t)$	–
P	Power	W
p	Semi-latus rectum	m
P_j	Exhaust jet power	W
ppr	Points per revolution	–
q	Exponent of highest order out-of-plane term for inverse polynomial method	–
q	Positive integer	–
$\dot{r}_f$	Radial velocity of arrival planet measured w.r.t. Sun	m/s
$\mathbf{r}$	Radial position vector	m
R	Matrix to store all cosine and sine terms for 2D FFS	–
r	Magnitude of position vector	m
$R(\theta)$	Function shaping r	m
$r(t)$	Fourier series for radius r	m
$r_{cp}(t)$	Cubic polynomial function for radius r	m
r_d	Distance between central body and perturbing body	m
r_f	Radius of arrival planet measured w.r.t. Sun	m
r_i	Radius of departure planet measured w.r.t. Sun	m
r_s	Distance between central body and spacecraft	m
$\dot{r}_i$	Radial velocity of departure planet measured w.r.t. Sun	m/s
s	Out-of-plane distance	m
T	Orbital period	s
T	Thrust	N
t	Time	s
$T(\theta)$	Time of flight function	s
$\mathbf{T_a}$	Thrust acceleration vector	m/s^2
T_a	Magnitude of thrust acceleration vector	m/s^2
$T_{a_{max}}$	Maximum allowed thrust acceleration	m/s^2
$T_{a_{r\theta}}$	In-plane thrust acceleration	m/s^2
T_{a_z}	Out-of-plane thrust acceleration	m/s^2
V	Velocity	m/s
$V_\theta(t)$	Transverse velocity function	m/s
V_e	Exhaust velocity	m/s
$V_r(t)$	Radial velocity function	m/s
$V_z(t)$	Axial velocity function	m/s
W	Available power	W
W	Energy flux	W/m^2
$[X_r]$	Fourier parameter $(a_n - f_n)$ vector	–
$\tilde{\mathbf{x}}$	State vector parametrised by azimuthal angle θ	m, m/s
$\mathbf{x}$	Decision vector containing unknown Fourier coefficients	–
x_i	Variable of the i-th dimension of Schwefel function	–

Greek Symbols

α	Right ascension	rad
α	Thrust pointing angle	rad
γ	Flight path angle	rad
ΔV	Change in velocity	m/s
δ	Declination	rad
ε	Ecliptic angle	rad
ε	Power conversion efficiency	–
ζ	Short-handed trigonometric estimation parameter	–
θ	Azimuthal angle	rad
θ	Polar angle	rad
θ	True anomaly	rad
θ_f	Transfer angle of arrival planet measured w.r.t. vernal equinox	rad
θ_i	Transfer angle of departure planet measured w.r.t. vernal equinox	rad
$\theta(t)$	Fourier series for transfer angle θ	rad

$\theta_{cp}(t)$	Cubic polynomial function for transfer angle θ	rad
$\dot{\theta}_f$	Angular velocity of arrival planet measured w.r.t. vernal equinox	rad/s
$\dot{\theta}_i$	Angular velocity of departure planet measured w.r.t. vernal equinox	rad/s
λ	Longitude	rad
μ	Gravitational parameter	$\mathrm{m^3/s^2}$
μ	Mean of normal distribution	–
σ	Standard deviation of normal distribution	–
σ_i	Slack variable	–
σ^2	Variance of normal distribution	–
τ	Scaled time	–
$\Phi(\theta)$	Function shaping ϕ	rad
ϕ	Elevation angle	rad
ϕ	Latitude	rad
ϕ	Phase shift of trigonometric initialisation function	rad
ϕ	Phasing angle	rad
Ω	Right ascension of the ascending node	rad
ω	Argument of pericenter	rad
ω	Frequency of trigonometric initialisation function	Hz
ω	Measure of the width of the tangent hyperbolic function	s/rad

Superscripts

$\dot{\square}$	First derivative with respect to time t
$\ddot{\square}$	Second derivative with respect to time t
$\square'$	First derivative with respect to azimuthal angle θ
$\square'$	First derivative with respect to scaled time τ
$\hat{\square}$	An integral with respect to polar angle
$\square''$	Second derivative with respect to azimuthal angle θ
$\square''$	Second derivative with respect to scaled time τ

Subscripts

[]

$\square_i$	Initial
$\square_f$	Final

Abbreviations

2PBVP	Two-Point Boundary Value Problem
AU	Astronomical Unit
CFF	Constrained Finite Fourier Series
CP	Cubic Polynomial
DP	Discretisation Point
DS1	Deep Space 1
DU	Distance unit
EoM	Equation(s) of Motion
FFS	Finite Fourier Series
GA	Genetic Algorithm
IP	Inverse Polynomial
JD	Julian Date
LEO	Low Earth Orbit
LTP	Low-Thrust Propulsion
MJD2000	Modified Julian Date, reference date: 01-01-2000. MJD2000 = JD - 2400000.5
NLP	Non-Linear Programming
NMP	New Millennium Program
SEP	Solar Electric Propulsion
SPICE	Spacecraft ephemeris - Planet, satellite, comet or asteroid ephemerides and physical, dynamical and cartographic constants - Instrument information - C-matrix orientation information - Events information
SQP	Sequential Quadratic Programming
SRP	Solar Radiation Pressure
TOF	Time Of Flight

TH	Tangent Hyperbolic
TU	Time unit
TU Delft	Delft University of Technology
TUDAT	Technical University of Delft Astrodynamics Toolbox
UFF	Unconstrained Finite Fourier Series

Table of Contents

1 Introduction

Because of its high efficiency, low-thrust propulsion is an attractive means to propel a spacecraft. Interplanetary transfer trajectories in particular benefit from low-thrust propulsion, considering the typically high ΔV to be achieved and the long transfer time required. In order to make the design of such missions fast, first-order, efficient representations of transfer orbits are commonly used before a more detailed and exact numerical model is applied. An elegant way to do so is to use so-called shape-based methods.

With Deep Space 1 being the first interplanetary low-thrust space mission in 1998 it did not take long before the scientific community picked up on the trajectory design of this new type of transfer orbits [16]. The exponential sinusoid by Petropoulos and Longuski [14] was one of the first shape-based methods. A more advanced method designed by Wall [27] makes use of an inverse polynomial and paved the way for other advanced methods like the spherical shaping method by Novak and Vasile [12], the hodographic shaping method by Gondelach [6] and the finite Fourier series method by Taheri and Abdelkhalik [21].

The latter method is capable of finding rendezvous trajectories that meet the initial and final position and velocity constraints, as well as the required time of flight. It can even incorporate a thrust magnitude constraint, which makes the found solutions more realistic and omits the need for a second optimisation run afterwards.

However, as with many shape-based methods, they become more fragile if the target body is at a higher inclination. Furthermore, Taheri and Abdelkhalik [22] explicitly state that the currently used initialisation strategy provides acceptable results, even though it might not be the best approach.

The purpose of this research is thence twofold: on the one hand it covers the implementation, validation and testing of the existing two-dimensional as well as the three-dimensional finite Fourier series shape-based method in TUDAT (Technical University of Delft Astrodynamics Toolbox). On the other hand, the goal is to introduce a new initialisation approach in order to decrease the solver convergence time and increase its three-dimensional stability, which can be translated to the following research question:

To what extent can an improved initialisation strategy enhance the efficiency and three-dimensional stability of the finite Fourier series method?

The document is divided in three parts and twelve chapters. Part I contains background information on the topic, starting with Chapter 2, which explains the relevant celestial mechanics. In Chapter 3 the potential of low-thrust propulsion and some examples are illustrated. Chapter 4 will then shortly introduce shape-based methods, after which a thorough explication of the finite Fourier series method in particular is given. The focus in Part II is on the implementation and validation of the finite Fourier series method. It includes a step-by-step guide of the two-dimensional and three-dimensional versions in Chapters 5 and 6. Subsequently, the model is validated against three cases described in literature in Chapter 7. Chapter 8 concludes this part with an overview of solved issues that were encountered during the implementation that have not been mentioned by literature. Part III involves the actual and novel research that has been performed on the existing finite Fourier series method, concentrating on the initialisation strategy in Chapter 9 in particular. Consecutively the results of the research in two and three dimensions are discussed in Chapters 10 and 11. Finally, conclusions of this research and recommendations for further research can be found in Chapter 12.

I

Background

2 Celestial Mechanics

In the field of astrodynamics, it is important to design the trajectory of a spacecraft while taking into account the right reference frame and assumptions. Information about the frames that are to be used, is found in Section 2.1. As soon as these are defined, the position of an object in space can be described by orbital elements, which is explained in Section 2.2. Followed by that are the different equations of motion in different coordinate systems in Section 2.3. To make appropriate assumptions, Section 2.4 concludes the chapter by discussing all perturbing forces acting on a spacecraft. Note that in Sections 2.1, 2.2 and 2.4 all material is based on Wakker [26], unless stated otherwise, and that only heliocentric orbits are considered.

2.1. Reference Frame

For interplanetary spaceflight it is most convenient to express the motion of a spacecraft in the non-rotating heliocentric ecliptic reference frame, or heliocentric reference frame in short. A graphical representation is shown in Figure 2.1. The reference frame is centred at the origin of the Sun and it uses the ecliptic plane as the reference plane. This plane is at an angle ε of $23°27'$ with respect to the equatorial plane of the Earth. The z-axis points towards the celestial north pole and the x-axis is fixed in the direction of the vernal equinox ♈. The direction of the y-axis is defined such that it lies in the reference plane and completes the right-handed reference frame.

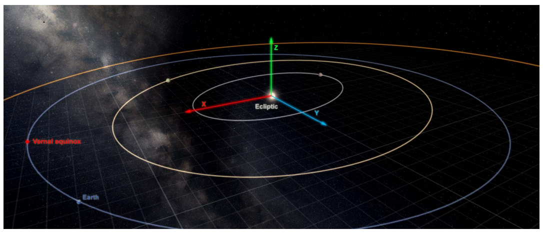


Figure 2.1: The axes of the heliocentric ecliptic reference frame [25].

2.2. Coordinates and Orbital Elements

To indicate the position of an object in space, the celestial sphere as shown in Figure 2.2 is used. In here, the declination δ is the angle measured from the plane of the celestial equator to the object S along an hour circle. The right ascension α is the angle measured from the vernal equinox to the projection of the object S along the celestial equator. The variables ϕ and λ represent the latitude and longitude and they are comparable to δ and α, respectively, but it should be kept in mind that the first two are measured with respect to the ecliptic. For these element, several formulations are available, of which the most general are referred to as Keplerian elements.

Another way of describing the location of an object in space is using so-called orbital elements. In two dimensions, a Kepler orbit can be described by three elements: the semi-major axis a, the eccentricity e and

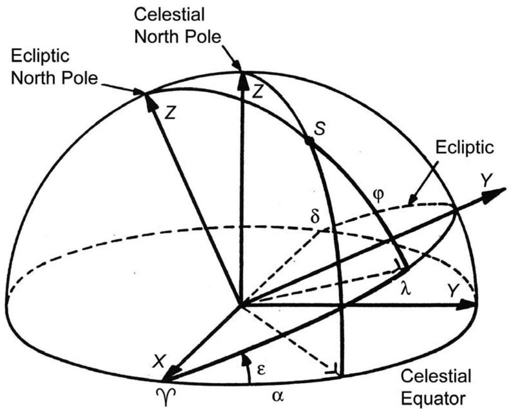


Figure 2.2: A depiction of the celestial sphere [26].

the true anomaly θ. The semi-major axis gives an indication of the size of the orbit. The shape of the orbit is expressed by the eccentricity. A value between zero and one shows how elliptic (e close to one), or how circular (e close to zero) the orbit is. Finally, the true anomaly is the angle between the line from the focus of the orbit towards the periapsis and the current position of the object.

If this is extended to three dimensions, some more elements are needed to fully describe the position of an object in space. The elements that are added are the inclination i, the right ascension (or longitude) of the ascending node Ω and the argument of pericenter ω. The inclination is the angle between the equatorial plane and the orbital plane. This quantity can never be smaller than 0° or larger than 180°. The right ascension of the ascending node goes by two names. In the equatorial plane, it is called the right ascension of the ascending node, while it is referred to as the longitude of the ascending node in the ecliptic plane. Both names represent the same parameter. It is the angle between the reference direction and the line that intersects the ascending node of the orbital plane, measured in the ecliptic or equatorial plane. Its value lies between 0° and 360°. The last important element is the argument of pericenter, which is the angle between a nodal line in a fixed reference direction and the radius vector from the origin to the pericenter. A clear overview is shown in Figure 2.3.

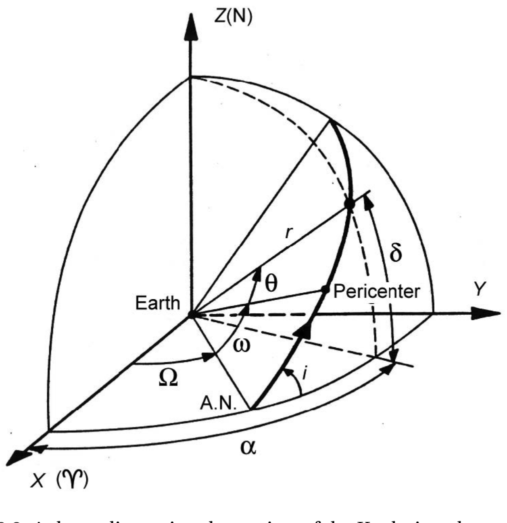


Figure 2.3: A three-dimensional overview of the Keplerian elements [26].

2.3. Equations of Motion

The equations of motion (EoM) describe the variation of the state of the spacecraft at any point in time and thus they are vital in getting insight in the dynamics of a trajectory. This section will briefly mention the three sets of equations of motion, relevant for various coordinates systems.

2.3.1. General Formulation

In an inertial frame and using cartesian coordinates, the EoM are given by the following equation, assuming two-body motion [26]:

$$\ddot{\mathbf{r}} + \frac{\mu}{r^3}\mathbf{r} = \mathbf{T_a} \tag{2.1}$$

The $\mathbf{T_a}$ vector represents the thrust acceleration, which is in fact a perturbation acting on the spacecraft. The position vector is denoted by $\mathbf{r}$, while the gravitational parameter is depicted by μ.

2.3.2. Polar Coordinates

For multiple shape-based methods however this is not the form that is generally used, since they often use polar coordinates. Therefore, Equation (2.1) has to be translated to such polar coordinates. In Figure 2.4 an overview of all vectors and angles are shown to describe the EoM in polar coordinates.

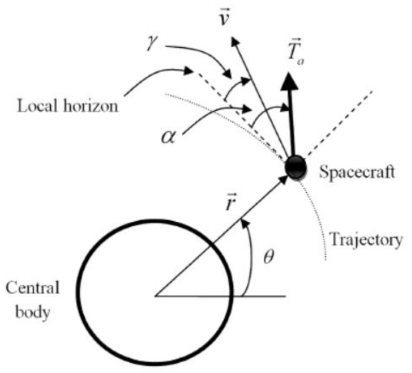


Figure 2.4: A representation of the variables used in the polar notation of the EoM [21].

The figure shows a spacecraft in orbit around a central body. The position with respect to the centre of the body is depicted by $\mathbf{r}$ and θ is the polar angle. The added angles are the flight path angle γ and the thrust pointing angle α. The flight path angle represents the angle between the local horizon and the velocity vector $\mathbf{v}$, while the thrust pointing angle is the angle between the thrust acceleration vector $\mathbf{T_a}$ and the local horizon. From the figure the following two equations can be derived [28]:

$$\begin{cases} \ddot{r} - r\dot{\theta}^2 + \frac{\mu}{r^2} = T_a \sin\alpha \\ 2\dot{r}\dot{\theta} + r\ddot{\theta} = T_a \cos\alpha \end{cases} \tag{2.2}$$

2.3.3. Cylindrical Coordinates

In case cylindrical coordinates are used, the equations mentioned above in polar form slightly change, because of the addition of a third component in the z-direction to describe the out-of-plane motion [27].

$$\begin{cases} \ddot{r} - r\dot{\theta}^2 + \frac{\mu}{s^3} r = T_{r\theta} \sin(\alpha) \\ 2\dot{r}\dot{\theta} + r\ddot{\theta} = T_{r\theta} \cos(\alpha) \\ \ddot{z} + \frac{\mu}{s^3} z = T_{a_z} \end{cases} \tag{2.3}$$

Note the difference in thrust acceleration with respect to the planar case. The subscript $r\theta$ in cylindrical coordinates denotes the acceleration in the two-dimensional xy-plane, whereas the subscript z indicates the acceleration in the z-direction. In Equation (2.3) the variable s is defined according to the following relation:

$$s = \sqrt{r^2 + z^2} \tag{2.4}$$

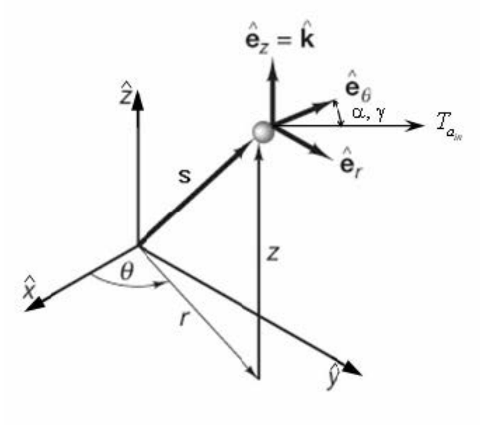
whereas it should be noted that the general expression for r is given by:

$$r = \sqrt{x^2 + y^2} \tag{2.5}$$

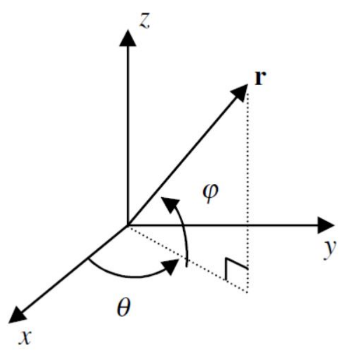
Figure 2.5a gives a graphical explanation of the cylindrical coordinate system that is used.

2.3.4. Spherical Coordinates

In addition to cylindrical coordinates, a spherical coordinate system can be used to express polar coordinates in three dimensions. The spherical coordinate system is defined in Figure 2.5b.



(a) A graphical representation of the cylindrical coordinate system [27].



(b) A graphical representation of the spherical coordinate system [12].

Figure 2.5: Two possible three-dimensional extensions of the polar coordinate system.

In here θ is the azimuthal angle, while ϕ is the elevation angle. The distance is described by $\mathbf{r}$ and is shown in Equation (2.6). Note that the direction of $\mathbf{r}$ is substantially different in the two coordinates sytems, as can be seen in Figure 2.5. In the cylindrical coordinate the radius vector is only expressed in the $xy-$plane, while it is extended in the $z-$direction as well in the spherical coordinate system.

$$\mathbf{r} = \begin{bmatrix} r\cos(\theta)\cos(\phi) & r\sin(\theta)\cos(\phi) & r\sin(\phi) \end{bmatrix}^T \tag{2.6}$$

The equations of motion for spherical coordinates are presented by Taheri et al. [23]:

$$\begin{bmatrix} \ddot{r} \\ \ddot{\theta} \\ \ddot{\phi} \end{bmatrix} = \begin{bmatrix} r\left(\dot{\theta}\cos(\phi)\right)^2 + r\dot{\phi}^2 - \frac{\mu}{r^2} \\ -2\frac{\dot{\theta}\dot{r}}{r} + 2\dot{\theta}\dot{\phi}\tan(\phi) \\ -2\frac{\dot{\phi}\dot{r}}{r} - \dot{\theta}^2\sin(\phi)\cos(\phi) \end{bmatrix} + \begin{bmatrix} 1 & 0 & 0 \\ 0 & \frac{1}{r\cos(\phi)} & 0 \\ 0 & 0 & \frac{1}{r} \end{bmatrix} \begin{bmatrix} f_r \\ f_\theta \\ f_\phi \end{bmatrix} \tag{2.7}$$

in which the thrust acceleration components are indicated by f. The subscript specifies their direction, which is either in the r-, θ- or ϕ-direction.

2.4. Perturbations

When designing a trajectory the effect of perturbations may have to be taken into account, but not taking them into account can greatly simplify the design process. In any case, a trade-off will need to be made, because if they are significant, not including them will not be beneficial for the accuracy of the results. An analysis of perturbations that are common for interplanetary trajectories is shown below in the following order: thrust, third-body perturbations, radiation pressure, gravity field perturbing forces, aerodynamic forces, electromagnetic forces and finally relativistic effects.

2.4.1. Thrust

Even though this might not be evident, a thrust force acting on a spacecraft is a perturbing force as well. For obvious reasons, low thrust is considered here only. When dealing with spacecraft that are equipped with low-thrust engines the following equation applies:

$$P_j = \varepsilon P = \frac{1}{2}\dot{m}V_e^2 = \frac{1}{2}FV_e \tag{2.8}$$

in which P is the electrical power that is generated by the solar panels, or even a nuclear reactor, P_j is the power of the exhaust jet, ε is the power conversion efficiency, $\dot{m}$ is the mass flow, F is the thrust force and V_e is the exhaust velocity of the engine. The thrust acceleration is simply found by dividing Equation (2.8) by the total mass M of the spacecraft. Two interesting conclusions can be drawn from this equation: first, in contrast to high thrust, the propulsive force generated by a low-thrust engine is not limited to the exhaust velocity, but to the maximum power. Secondly, the thrust force decreases with an increasing exhaust velocity. The range of the thrust accelerations that a low-thrust engine can provide can be seen in Figure 2.6. Because it is the propulsive force of the spacecraft, it cannot be neglected.

2.4.2. Third-Body Perturbations

Third-body perturbations are caused by the influence of other celestial bodies and are thus gravitational. In the reader by Wakker [26] an equation for the relative perturbing acceleration of a spacecraft is given:

$$\left(\frac{a_d}{a_k}\right)_{max} = \frac{m_d}{m_k}\left(\frac{r_s}{r_d}\right)^2 \left|\left(\frac{1}{1-\frac{r_s}{r_d}}\right)^2 - 1\right| \tag{2.9}$$

The equation shows the ratio of the acceleration a_d of perturbing body d over the main acceleration a_k of body k on spacecraft s. In case of an interplanetary flight, body k is the Sun, which reduces the influence of these perturbations to a marginal level, as the mass fraction $\frac{m_d}{m_k}$ becomes very small. However, the term between brackets can become significant in case the spacecraft comes close to the perturbing body. This effect is nicely depicted in Figure 2.6. From here it can be concluded that it can be left out of the models, unless use is made of a flyby.

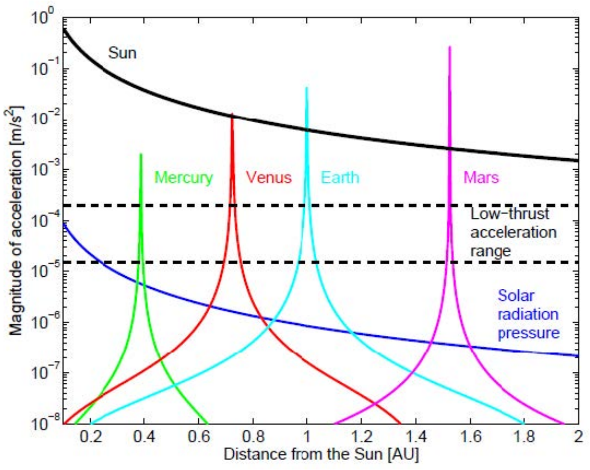


Figure 2.6: An overview of the magnitude of accelerations due to the gravity, a thrust force and Solar radiation pressure [6]. This spacecraft has a cross-sectional area of 40 m^2, a mass of 400 kg and a coefficient of reflectivity of 1.9.

2.4.3. Radiation Pressure

When talking about radiation pressure, it is mainly solar radiation pressure (SRP). There are other types of radiation as well, like albedo or infra-red radiation, but for an interplanetary spacecraft, these barely influence the accelerations acting on the spacecraft. Therefore, the radiation pressure is specified to SRP only, which allows the acceleration caused by this radiation to be described by the following equation:

$$\mathbf{f}_{SRP} = -C_R \frac{WA}{mc}\, \hat{\mathbf{e}}_{\mathbf{S}} \tag{2.10}$$

In here, $\mathbf{f}_{SRP}$ is the acceleration caused by the radiation, C_R is the reflectivity of the spacecraft, W is the energy flux of the incoming Solar radiation, A is the cross-sectional area of the spacecraft, m is its mass and c is the speed of light. The vector $\mathbf{e}_{\mathbf{S}}$ is the unit vector from the spacecraft to the Sun, and SRP is always pointing in the opposite direction of the Sun, hence the minus sign.

For a spacecraft with an $\frac{A}{m}$ ratio of 12 m^2/kg and a C_R of 1.9 in an orbit around the Earth, the acceleration is only 0.10 mm/s^2 [26]. For an interplanetary flight to the outer planets of the Solar System, this can only decrease, as the energy flux W decreases quadratically with the distance from the Sun. This is visible in Figure 2.6. From here it can be seen that it can be easily neglected, unless a trajectory to Mercury is flown.

2.4.4. Gravity Field Perturbing Forces

None of the celestial bodies is a perfect sphere. There always exist some deviations and anomalies in the gravity field. Also, most planets are more like an oblate sphere. This all influences the gravity force at a certain point in the orbit of the spacecraft. The effects of these imperfections are expressed in e.g. J_2 or $J_{2,2}$ terms for a spherical harmonics gravity model. For interplanetary flight, these perturbations will only be considered in case of a flyby, as Figure 2.6 shows that gravity forces are only of a significant magnitude when the spacecraft is close to such a body.

2.4.5. Aerodynamic Forces

A spacecraft can only experience aerodynamic forces when it is flying at low altitude and an atmosphere is present on the nearby planet. As the drag is a function of the density, which on itself is a function of the altitude, the effects of this perturbing force quickly reduce at high altitude. During an interplanetary trajectory the spacecraft usually does not cross the atmosphere of a planet. For that reason, it is concluded that it can be neglected.

2.4.6. Electromagnetic Forces

In the intense radiation environment of outer space, a spacecraft is constantly hit by highly charged particles. These cause the spacecraft to have a potential difference over the entire object. With the magnetic field of the Earth and a little amount of current flowing through the spacecraft, it will experience a Lorentz force. The magnetic field decreases in strength with increasing altitude. A spacecraft with an orbital-radius-over-mass-ratio of 0.3 m/kg at an altitude of 500 km experiences an acceleration of only 6.0×10^{-10} m/s^2 [26]. Knowing that this is only for a spacecraft in a low Earth-orbit (LEO), it is definitely insignificant for interplanetary travel.

2.4.7. Relativistic Effects

Finally, there are relativistic effects. Relativity can cause disturbances in the modelling of an orbit as well. It is responsible for the precession of the argument of periapsis, for example. Nonetheless, these effects are only very marginal and in an order of time of one century, hence, it can be neglected for interplanetary trajectory design.

2.4.8. Summary

From the seven perturbing forces that are mentioned in Sections 2.4.1 to 2.4.7, thrust cannot be neglected, while third-body perturbations and solar radiation pressure are only relevant to take into account for interplanetary flight, provided that the spacecraft is close to those bodies, e.g. in case of a gravity-assist. This is only true for a minor part of the trajectory and therefore not of prime importance. The others can be neglected in any case, or they are simply not applicable. Aerodynamic perturbations for example only need to be considered in case of aerogravity-assists. This implies that an interplanetary low-thrust orbit can be accurately modelled using the two-body approximation with thrust as the only extra component.

3 Low-Thrust Propulsion

Trajectory design has been a fascinating topic even before technology allowed the first objects to actually reach outer space. When that finally happened, developments in this new field of engineering skyrocketed and eventually led to the design of extremely efficient low-thrust engines. In this chapter, a succinct survey of low-thrust propulsion (LTP) and its incidentals is given, starting with the principles of LTP in Section 3.1. The different types of LTP are addressed in Section 3.2. Furthermore, two of the very first low-thrust space missions and their newly discovered advantages are reviewed in Section 3.3. Finally, the new way of trajectory design that comes with this type of trajectories is shortly touched upon in Section 3.4.

3.1. Physical Principles

The thrust generated by a rocket engine is a function of the mass flow $\dot{m}$ and the exhaust velocity V_e of the propellant:

$$T = \dot{m} \cdot V_e \tag{3.1}$$

The thrust relies on Newton's famous third law stating that for every action, there is an equal and opposite reaction. This means that the thrust force T that expels the particles also acts on the spacecraft. The energy that is given to the ejected propellant can be expressed as a change of kinetic energy per time unit:

$$\frac{\mathrm{d}E}{\mathrm{d}t} = W = \frac{\mathrm{d}}{\mathrm{d}t}\left(\frac{mV_e^2}{2}\right) = \frac{\dot{m}V_e^2}{2} \tag{3.2}$$

in which W is the power that is stored in the ejected gas. With this information, Equation (3.1) can be rewritten as follows:

$$T = \sqrt{2\dot{m}W} = \frac{2W}{V_e} \tag{3.3}$$

Both Equations (3.2) and (3.3) show interesting insights in the effect of the available power and exhaust velocity on the generated thrust. Equation (3.2) states that with a fixed thrust level the propellant used is minimised at a high exhaust velocity, while on the contrary, Equation (3.3) shows that with a fixed amount of available power, a high exhaust velocity leads to a lower thrust [19].

Another important parameter that is used to determine the thrust characteristics of an engine, is the specific impulse I_{sp}. This unit is a measure of the efficiency of a rocket engine and it is defined as follows:

$$I_{sp} = \frac{T}{\dot{m}g_0} = \frac{V_e}{g_0} \tag{3.4}$$

in which g_0 is the standard Earth gravity of 9.81 m/s^2. In Table 3.1 an overview of several solar electric propulsion (SEP) engines and their specific impulse ranges can be found. Whereas conventional chemical engines have a specific impulse in the order of 220 to 400 s, the table clearly shows the higher order ranges for ion engines [30].

Table 3.1: An overview of different SEP engines and their characteristics [8].

Engine	T6	UK-25	RIT-XT	RIT-35		ESA-XX
Exhaust gas	Xe	Xe	Xe	Xe	Hg	Xe
Mass [kg]	7.5	9	7	9	9	-
Power range [kW]	1.3 - 7.0	0.1 - 8	1.4 - 5.6	1.5 - 7	1 - 18	? - 8.45
Specific impulse [s]	3000 - 4500	2800 - 4700	2100 - 5500	up to 5000	up to 4050	? - 5400
Thrust range [mN]	30 - 205	37 - 320	15 - 200	40 - 200	50 - 250	10 - 240

3.2. Types of Low-Thrust Propulsion

In general, three different types of LTP engines can be distinguished: electrostatic engines, electrothermal engines and electromagnetic engines. All will be shortly introduced in the upcoming three subsections.

Electrostatic Engines

Electrostatic engines use electrostatic forces to accelerate the particles in their exhaust gas. By means of electricity the engine ionises the gas, after which a voltage gradient is accelerated to accelerate the particles to a high exhaust velocity. Usually spacecraft that operate an engine of this type are equipped with solar panels, but in case of missions to the outer Solar System or when simply a large amount of power is required, the energy might come from a nuclear power source [6].

Electrothermal Engines

Electrothermal thrusters generate electromagnetic fields that are used to create a plasma in order to heat the propellant. The thermal energy which is then stored in this propellant is converted into kinetic energy, which results in a high exhaust velocity of the particles [6].

Electromagnetic Engines

Electromagnetic thrusters can accelerate the ions in two ways: either by creating a magnetic field that induces the Lorentz force, or by creating an electric field and making use of the Coulomb force [6].

3.3. Low-Thrust Space Missions

The first spacecraft propelled by a low-thrust engine only flew in 1998. This section highlights and points out the newly-discovered possibilities of two missions by the two most notable space agencies: Deep Space 1 by NASA, and SMART-1 by ESA.

Deep Space 1

Launched in 1998, Deep Space 1 (DS1) was the first mission developed during the New Millennium Program (NMP) by NASA. It was revolutionary in a sense that it made use of low-thrust SEP. The use of SEP for this mission had three objectives: it should demonstrate flight operations using this new technology, it should assess its interaction with the spacecraft and it should be capable of propelling the spacecraft into the desired trajectory during which it would encounter an asteroid, as seen in Figure 3.1 [16].

In the past, these kind of missions were only studied at the level of conceptual design. With DS1 demanding to dive into the design with much more detail, other interesting facets of LTP were found. Continuous thrust is to be used in a direction tangential to the trajectory, for example. This comes with constraints on the attitude or communication, because the antenna would not always be pointing towards Earth.

In the end the mission turned out to be a successful venture and a large amount of new knowledge regarding low-thrust trajectory and spacecraft design was gained, as the mission fulfilled all three objectives the NMP stated well [16].

SMART-1

The ESA version of the NMP is called the ESA Horizons 2000 scientific programme. It launched their first mission only a few years after DS1 in 2003 and it was called SMART-1. This was a mission to the Moon and again, it would primarily be used as a test for new technology and the possibilities of SEP in particular. It would take approximately 15 to 18 months before it was injected into a Lunar orbit. A benefit of SEP in this

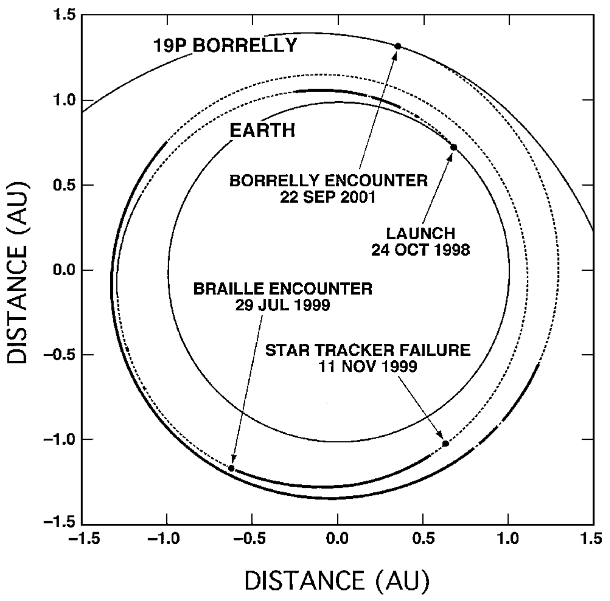


Figure 3.1: The trajectory of the Deep Space 1 spacecraft [16].

case, for example, is that it offered enough time to calibrate the instruments and even to do some additional measurements before reaching its final destination [4].

3.4. Modelling Low-Thrust Transfer Trajectories

Modelling and optimising these previously discussed mission trajectories asks for a completely different approach compared to high-thrust trajectories, but the type of problem is essentially the same and can be solved by two different methods: direct and indirect ones. Both numerical methods have their advantages and disadvantages.

When an indirect method is used, the trajectory is found by solving a control problem that corresponds to a so-called two-point boundary value problem (2PBVP). This method can be solved by giving an initial guess as input. A defect of this method is its high sensitivity to this initial solution. It easily heads towards the wrong direction if this guess is of bad quality. However, the convenience of this method is found in the solution of the 2PBVP, because the resulting trajectory determined thence is often optimal.

Direct methods on the other hand do not make use of a 2PBVP and directly solve the control problem. By adjusting the control variables during each loop the algorithm eventually converges to the desired trajectory. For direct methods the initial guess can be more robust. A downside of direct methods is that they are slower compared to indirect methods [7].

4 Shape-Based Methods

Having addressed the capabilities of low-thrust propulsion, the actual trajectory design methods, shape-based methods in particular, can be addressed. In Section 4.1, a short introduction of the most common current shape-based methods can be read. Furthermore, a comparison of the properties of these methods is given in Section 4.2. After that, a profound explanation of the finite Fourier series method is presented in Section 4.3.

4.1. Introduction

If low-thrust propulsion is used, the assumption of an impulsive shot that applies to general high-thrust trajectory design does not hold anymore. This new type of propulsion requires a new trajectory design approach, which is best represented by shape-based methods. The theory behind these methods relies on the fact that the (usually spiralling) trajectories are described by a shaping function that assumes a certain shape which the trajectory is fitted to. In this section, a short overview of the most common shape-based methods and their governing shaping functions are given. A more elaborate explication can be found in Appendix F.

One of the very first shape-based methods is the exponential sinusoid (exposin) by Petropoulos and Longuski [14]. It is described by Equation (4.1) and relies on the constants k_0, k_1, k_2 and ϕ. The winding parameter k_2 determines the number of revolutions, the dynamic range parameter k_1 controls the ratio of the apocenter distance to the pericenter distance, the parameter k_0 is only a scaling factor and the phase angle ϕ controls the orientation of the exponential sinusoid in the plane.

$$r = k_0 \mathrm{e}^{(k_1 \sin(k_2\theta+\phi))} \tag{4.1}$$

Later on, Wall and Conway [28] designed the inverse polynomial method, which was inspired by the exposin. It expresses the shape of the transfer orbit by a fifth or sixth-order polynomial, as can be seen in Equation (4.2). The order depends on the time of flight (TOF). If it is free, a fifth-order polynomial can be used, but when it is fixed, a sixth-order polynomial is required in order to include it properly. The coefficients a to f are then found by inserting the boundary conditions (BC) in the equation while assuming tangential thrust.

$$r(\theta) = \frac{1}{a + b\theta + c\theta^2 + d\theta^3 + e\theta^4 + f\theta^5 + g\theta^6} \tag{4.2}$$

The spherical shaping method by Novak and Vasile [12] is also parametrised by θ instead of t, just like the inverse polynomial method. It is defined in the spherical coordinate system and uses a shaping function for the radius $R(\theta)$, the elevation angle $\Phi(\theta)$ and the TOF $T(\theta)$. It is the first fully three-dimensional method and it is especially well suited for modelling trajectories to targets at higher inclinations. The shaping coefficients are denoted by a_n and b_n, while $D(\theta)$ is the time equation scalar function that ensures the curvature of the trajectory is towards the target body. Finally, μ describes the gravitational parameter of the central body.

$$R(\theta) = \frac{1}{a_0 + a_1\theta + a_2\theta^2 + (a_3 + a_4\theta)\cos(\theta) + (a_5 + a_6\theta)\sin(\theta)} \tag{4.3a}$$

$$\Phi(\theta) = (b_0 + b_1\theta)\cos(\theta) + (b_2 + b_3)\sin(\theta) \tag{4.3b}$$

$$T(\theta)' = \sqrt{\frac{D(\theta)\,R(\theta)^2}{\mu}} \tag{4.3c}$$

The hodographic shaping method by Gondelach [6] is originated at TU Delft and shapes the trajectories in the velocity domain, rather than the spatial domain. This means that there are three shaping functions: the radial velocity V_r, the transverse velocity V_θ and the axial velocity V_z. Each of them is set up as in Equation (4.4), which entails that it is composed of a sum of coefficients c_i and velocity functions v_i. The latter can have any shape from a second-order polynomial to an exponential sinusoid. Furthermore, the method can both be parametrised by the transfer angle θ and the time t.

$$V(t) = \sum_{i=1}^{n} c_i v_i(t) \tag{4.4}$$

Back in the spatial domain and the time domain, Taheri and Abdelkhalik [21] designed a method that approximates trajectories by means of Fourier series. The three components are shown in Equation (4.5). As this method is the main focus of this thesis, a thorough explanation can be found later on in Section 4.3.

$$r(\tau) = \frac{a_0}{2} + \sum_{n=1}^{n_r} \{a_n \cos(n\pi\tau) + b_n \sin(n\pi\tau)\} \tag{4.5a}$$

$$\theta(\tau) = \frac{c_0}{2} + \sum_{n=1}^{n_\theta} \{c_n \cos(n\pi\tau) + d_n \sin(n\pi\tau)\} \tag{4.5b}$$

$$z(\tau) = \frac{e_0}{2} + \sum_{n=1}^{n_z} \left\{e_n \cos(n\pi\tau) + f_n \sin(n\pi\tau)\right\} \tag{4.5c}$$

The final method to be discussed is the finite polynomil method. Its shaping function in Equation (4.6) is very similar to the inverse polynomial method, but here the radial distance and the transfer angle are shaped by two different functions that are both parametrised by time.

$$r(t) = a_{r_0} + a_{r_1} t + a_{r_2} t^2 + a_{r_3} t^3 + \ldots + a_{r_m} t^m \tag{4.6a}$$

$$\theta(t) = a_{\theta_0} + a_{\theta_1} t + a_{\theta_2} t^2 + a_{\theta_3} t^3 + \ldots + ... + a_{\theta_n} t^n \tag{4.6b}$$

4.2. Comparison

Knowing the ins and outs of various shape-based methods, it is possible to make a clear comparison between them to highlight their strengths and weaknesses. This overview is captured in Table 4.1. Green cells show properties that are beneficial, as these might improve the accuracy, feasibility and flexibility of the generated trajectories. Red cells on the other hand indicate unwanted properties, as they could lead to long computation times, a worse accuracy or infeasible trajectories.

From the table it can be extracted that all methods have BCs set on the position, the velocity and the TOF. The exposin is the only method that cannot handle a BC on its velocity, whereas the finite Fourier series and the finite polynomial do not have a BC on the TOF. For these two methods, this is just set by trial and error. The inverse polynomial and the spherical shaping method on the other hand do have a BC on the TOF, but this cannot be solved directly. It has to be found by means of an iterative root-finding algorithm.

Also, almost all methods work with inclined trajectories, except for the exposin, as this is only capable of marginal plane changes. Most of the methods that are said to work in three dimensions, provide acceptable results up to an inclination of 15°, like the inverse and the finite polynomial. For the finite Fourier transform it is not exactly known what the limit is, but the highest achieved inclination that has been found in literature is that of asteroid Dionysus, which is 13.6°. By far, it is the spherical shaping method that performs notably

Table 4.1: An overview of the most important capabilities of a shape-based method, partially taken from Gondelach [6] and Roegiers [17].

Method	BC			BC solved iteratively	3D	Thrust constraint	≥ 2 Revolutions
	r	**V**	TOF				
Exposin	Yes	No	Yes	No	No	No	Yes
Inverse polynomial	Yes	Yes	Yes	Yes	Yes	No	No
Spherical shaping	Yes	Yes	Yes	Yes	Yes	No	Yes
Hodographic shaping	Yes	Yes	Yes	No	Yes	No	No
Finite Fourier series	Yes	Yes	No	No	Yes	Yes	Yes
Finite polynomial	Yes	Yes	No	No	Yes	Yes	No

well. With the original design by Novak and Vasile [12], it was already capable of attaining an inclination of 50°, but with the new shaping function by Vroom [25] it could reach objects with an inclination of 70°.

Furthermore, for some methods a constraint can be set on the thrust, or thrust acceleration. If this is already implemented in the method, it is beneficial considering computation time. The other methods can work with a thrust constraint as well, but for those it will be part of the optimisation process and it is not directly incorporated in the method itself, which is computationally intense. Only the finite Fourier series method and the finite polynomial have this constraint implemented, which is a major advantage, as it will give a mission planner a much clearer overview due to infeasible trajectories that would require a thrust that is technologically not feasible being immediately discarded in the design process.

Lastly, the capable number of revolutions need to be discussed. It is only the inverse and the finite polynomial that seem to have trouble in case the trajectory contains two or more revolutions. This has probably to do with the shaping functions that they use, as all the other methods have (the possibility) to use periodic functions. The two polynomial functions are not capable of doing so, consequently leading to bad results when many revolutions are considered.

4.3. Finite Fourier Series

The Finite Fourier Series (FFS) method is the main focus of this thesis and therefore it is awarded with its own chapter. First, the method in two dimensions is explained in Section 4.3.1, after which its advantageous extension in three dimensions is elaborated upon in Section 4.3.2.

4.3.1. 2D Finite Fourier Series

The method of FFS was first described by Taheri and Abdelkhalik [21]. It does not directly generate a fixed shape, but it rather assumes an approximation for the shape in terms of an FFS. Using an FFS, one can choose between two possible approaches: in one approach a radius r is assumed as a function of the polar angle θ, which is then expanded. Taheri and Abdelkhalik [21] however decided to follow the other approach, which assumed two separate expressions for r and θ as a function of time. These are shown below:

$$r(t) = \frac{a_0}{2} + \left\{ \sum_{n=1}^{n_r} a_n \cos\left(\frac{n\pi}{T} t\right) + b_n \sin\left(\frac{n\pi}{T} t\right) \right\} \tag{4.7a}$$

$$\theta(t) = \frac{c_0}{2} + \left\{ \sum_{n=1}^{n_\theta} c_n \cos\left(\frac{n\pi}{T} t\right) + d_n \sin\left(\frac{n\pi}{T} t\right) \right\} \tag{4.7b}$$

In here, T is the total TOF, and n_r and n_θ indicate the number of Fourier terms that will be included. This depends on the type of trajectory that is flown, but both of them being at least two, all BCs can be satisfied in case of a rendez-vous.

Mathematical Fundamentals

The FFS assumes that the thrust direction is always pointing in the direction of the velocity, or against it, so α equals γ (recall Figure 2.4). This condition will be used when the EoM in polar form are combined into one expression. Both equations are stated here again for the sake of clarity:

$$\alpha = \gamma + n\pi \tag{4.8}$$

$$\begin{cases} \ddot{r} - r\dot{\theta}^2 + \frac{\mu}{r^2} = T_a \sin\alpha \\ 2\dot{r}\dot{\theta} + r\ddot{\theta} = T_a \cos\alpha \end{cases} \tag{4.9}$$

Rewriting the second equation of Equation (4.9) for T_a results in the following expression:

$$T_a = \frac{2\dot{r}\dot{\theta} + r\ddot{\theta}}{\cos(\alpha)} \tag{4.10}$$

Substituting this expression for T_a into the first equation in Equation (4.9), yields:

$$\ddot{r} - r\dot{\theta}^2 + \frac{\mu}{r^2} = \frac{2\dot{r}\dot{\theta} + r\ddot{\theta}}{\cos(\alpha)} \sin(\alpha) = \left(2\dot{r}\dot{\theta} + r\ddot{\theta}\right)\tan(\alpha) = \left(2\dot{r}\dot{\theta} + r\ddot{\theta}\right)\tan\left(\gamma\right) \tag{4.11}$$

With the assumption of tangential thrust, the tangent can be replaced by:

$$\tan(\alpha) = \tan\left(\gamma\right) = \frac{\dot{r}}{r\dot{\theta}} \tag{4.12}$$

If then Equation (4.12) is plugged into Equation (4.11), some algebraic manipulation is needed to finally obtain one governing equation:

$$f\left(r, \dot{r}, \ddot{r}, \dot{\theta}, \ddot{\theta}\right) = r^2\left(\dot{\theta}\ddot{r} - \dot{r}\ddot{\theta}\right) + \dot{\theta}\left(\mu - 2r\dot{r}^2\right) - \left(r\dot{\theta}\right)^3 = 0 \tag{4.13}$$

The following condition follows directly from the imposed relation in Equation (4.8).

$$\cos(\alpha) = \cos\left(\gamma\right) = \frac{r\dot{\theta}}{\sqrt{\dot{r}^2 + \left(r\dot{\theta}\right)^2}} \tag{4.14}$$

The thrust acceleration can then be calculated from Equation (4.10) and Equation (4.8). Note that C expresses the entire constraint equation as in:

$$C : \left(\frac{T_a}{T_{a_{max}}}\right)^2 \leq 1 \tag{4.15}$$

If the two expressions in Equations (4.7a) and (4.7b) and their first and second derivaties are inserted in Equation (4.13), the initial differential equation is transformed into a non-linear algebraic expression in which the Fourier coefficients and the independent time variable are the only unknown terms:

$$f\left(a_0, a_1, \cdots, a_{n_r}, b_1, \cdots, b_{n_r}, c_0, c_1, \cdots, c_{n_\theta}, d_1, \cdots, d_{n_\theta}; t\right) = 0 \tag{4.16}$$

General Approach

To find the trajectory, the Fourier coefficients need to be found. Depending on n_r and n_θ there are $n = 2(n_r + n_\theta + 1)$ unknown coefficients to be solved for. The total TOF is divided by m discretisation points (DP), which creates a non-linear system of m equations at m moments in time.

The general approach can be explained as follows: given a departure time, a TOF, N_{rev}, n_r and n_θ: first the boundary values (initial and final (angular) position and (angular) velocity) are computed. Followed by that, an initial guess for the unknown coefficients is computed. The third step is to solve for the first eight coefficients by enforcing the BCs. After that, the total TOF needs to be discretised into intervals and evaluated at their boundary points to create m equations. Possibly, in case the thrust constraint is included in the system, the total number of equations sums up to $2m$ equations, of which half consists of inequalities. Now the non-linear programming problem can be solved at each time step, using the previous result as an initial guess for the current time. This solving procedure can be formulated as an optimisation problem of the following form:

$$\begin{aligned} \min \quad & \left(\sum_{n=1}^{m} f\left(a_0, a_1, \cdots, a_{n_r}, b_1, \cdots, b_{n_r}, c_0, c_1, \cdots, c_{n_\theta}, d_1, \cdots, d_{n_\theta}; t_n\right)\right)^2 \\ \text{s.t.} \quad & \left(\frac{T_a}{T_{a_{max}}}\right)^2 \leq 1 \end{aligned} \tag{4.17}$$

Initial Guess

As the problem is non-linear, the system requires an initial guess for the coefficients. This is done by assuming a simple shape for the trajectory, from which the corresponding Fourier coefficients are found to be used as a priori information about the possible solution. For orbit-changing problems, use could be made of two methods: a tangent hyperbolic (TH), or the cubic polynomial (CP). The TH is defined according to:

$$r(t) = \frac{1}{2}\left[(a_r + b_r) + (b_r - a_r)\tanh\left(\frac{t - t_0}{\omega}\right)\right] \tag{4.18a}$$

$$\theta(t) = \frac{1}{2}\left[(a_\theta + b_\theta) + (b_\theta - a_\theta)\tanh\left(\frac{t - t_0}{\omega}\right)\right] \tag{4.18b}$$

where a_r and a_θ are equal to r_0 and θ_0, while b_r and b_θ are equal to r_f and θ_f. t_0 is $T/2$ and ω is an indicator for the width of the function. It can be scaled such that it provides a gradual increase in the two parameters.

The CP approximation is defined by:

$$r_{CP}(t) = at^3 + bt^2 + ct + d \tag{4.19a}$$

$$\theta_{CP}(t) = et^3 + ft^2 + gt + h \tag{4.19b}$$

where the eight BCs are used to compute all coefficients. With Equations (4.7a) and (4.19a) a linear system of the form $A\mathbf{x} = B$ can be set up, in which the Fourier coefficient vector $\mathbf{x}$ in Equation (4.20) is found by left-multiplying both sides with A^{-1}.

$$\mathbf{x} = \begin{bmatrix} a_0 & a_1 & a_2 & \cdots & a_{n_r} & \cdots & b_1 & b_2 & \cdots & b_{n_r} \end{bmatrix}^T \tag{4.20}$$

In this expression the trigonometric terms from Equation (4.7a) are collected in A, which leads to Equation (4.21):

$$A = \begin{bmatrix} \frac{1}{2} & \cos\left(\frac{\pi}{T}t_0\right) & \cos\left(\frac{2\pi}{T}t_0\right) & \cdots & \cos\left(\frac{n_r\pi}{T}t_0\right) & \sin\left(\frac{\pi}{T}t_0\right) & \sin\left(\frac{2\pi}{T}t_0\right) & \cdots & \sin\left(\frac{n_r\pi}{T}t_0\right) \\ \frac{1}{2} & \cos\left(\frac{\pi}{T}t_1\right) & \cos\left(\frac{2\pi}{T}t_1\right) & \cdots & \cos\left(\frac{n_r\pi}{T}t_1\right) & \sin\left(\frac{\pi}{T}t_1\right) & \sin\left(\frac{2\pi}{T}t_1\right) & \cdots & \sin\left(\frac{n_r\pi}{T}t_1\right) \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ \frac{1}{2} & \cos\left(\frac{\pi}{T}t_i\right) & \cos\left(\frac{2\pi}{T}t_i\right) & \cdots & \cos\left(\frac{n_r\pi}{T}t_i\right) & \sin\left(\frac{\pi}{T}t_i\right) & \sin\left(\frac{2\pi}{T}t_i\right) & \cdots & \sin\left(\frac{n_r\pi}{T}t_i\right) \end{bmatrix} \tag{4.21}$$

This then becomes a matrix of $(2n_r + 1)$ rows, as the total TOF is divided into $(2n_r + 1)$ intervals, and it has $(2n_r + 1)$ columns. Its size is thus fully depending on the number of coefficients that are used. The B matrix is a column vector that contains the evaluation of Equation (4.19a) at $(2n_r + 1)$ points in time, which can be seen in Equation (4.22):

$$B = \begin{bmatrix} r_{CP}(t_0) & r_{CP}(t_1) & \cdots & r_{CP}(t_i) \end{bmatrix}^T \tag{4.22}$$

To find the c_n and d_n Fourier coefficients, the same procedure is followed, but this time Equations (4.7b) and (4.19b) are used.

It should however be noted that in contrast to what is stated in both the paper by Taheri and Abdelkhalik [21], and the PhD thesis by Taheri [20], the time discretisation method stated above is slightly different, as the two scientific documents both mention different matrix dimensions. It is written that Equations (4.7a) and (4.19a) are evaluated at n_r points in time, which means the column dimension of the matrix can never exceed n_r. Yet, it also mentions that matrix A should be an $(2n_r + 1)$ by $(2n_r + 1)$ matrix, while matrix B is a $(2n_r + 1)$ by 1 matrix. This last statement is simply not possible and therefore the time has to be discretised into $(2n_r + 1)$ intervals.

4.3.2. 3D Finite Fourier Series

Just like the inverse polynomial method, Taheri and Abdelkhalik [22] extended the FFS approach into the third dimension by adding a component in the z-direction and thereby adopting the cylindrical coordinate system. The approach for determining the coefficients is largely the same as in the two-dimensional case. Moreover, the new variable τ is introduced, which scales the total TOF according to $\tau = \frac{t}{T}$ onto the interval $[0,1]$. The three position components are defined as follows:

$$r(\tau) = \frac{a_0}{2} + \sum_{n=1}^{n_r} \{a_n \cos(n\pi\tau) + b_n \sin(n\pi\tau)\} \tag{4.23a}$$

$$\theta(\tau) = \frac{c_0}{2} + \sum_{n=1}^{n_\theta} \{c_n \cos(n\pi\tau) + d_n \sin(n\pi\tau)\} \tag{4.23b}$$

$$z(\tau) = \frac{e_0}{2} + \sum_{n=1}^{n_z} \{e_n \cos(n\pi\tau) + f_n \sin(n\pi\tau)\} \tag{4.23c}$$

Mathematical Fundamentals

With the adoption of the new cylindrical coordinate system comes a redefinition of the equations of motion from Equation (4.9) as well:

$$\begin{cases} \ddot{r} - r\dot{\theta}^2 + \frac{\mu}{s^3} r = f_r \\ 2\dot{r}\dot{\theta} + r\ddot{\theta} = f_\theta \\ \ddot{z} + \frac{\mu}{s^3} z = f_z \end{cases} \tag{4.24}$$

It can be seen that the z-component is added and that the variable s has been introduced, affecting the r and θ-components too. The variable s is found by applying the Pythagorean theorem to the radial distance r and the height z, leading to:

$$s = \sqrt{r^2 + z^2} \tag{4.25}$$

The right-hand side of Equation (4.24) also introduces three new variables f_r, f_θ and f_z, which represent the thrust acceleration components in the radial, transverse and axial direction, respectively. The total thrust acceleration T_a is then found by:

$$T_a = \sqrt{f_r^2 + f_\theta^2 + f_z^2} \tag{4.26}$$

from which the total ΔV can be computed by integrating T_a according to:

$$\Delta V = \int_0^T T_a \, \mathrm{d}t \tag{4.27}$$

With 12 BCs (an initial and final condition for each variable in terms of position and velocity) Taheri and Abdelkhalik [22] decided that the first two coefficients of each Fourier approximation, excluding the first constant number, will have to be expressed in terms of these BCs, as these coefficients influence the convergence of the solver more than lower-order terms [24]. This implies that the individual number of coefficients, i.e. n_r, n_θ and n_z will each need to be larger than or equal to two. The BCs are defined as follows:

$$\begin{aligned} r_i &= r_i & r_f &= r_f \\ r_i' &= T\dot{r}_i & r_f' &= \dot{r}_f \\ \theta_i &= \theta_i & \theta_f &= \theta_f \\ \theta_i' &= T\dot{\theta}_i & \theta_f' &= T\dot{\theta}_f \\ z_i &= z_i & z_f &= z_f \\ z_i' &= T\dot{z}_i & z_f' &= T\dot{z}_f \end{aligned} \tag{4.28}$$

The BCs imposed on r and θ are identical to the ones used for the two-dimensional approach. The BCs that are set on z indicate the axial position, which is effectively the "height" in the Solar System, as seen from the ecliptic. It is also important to observe that due to the introduction of the scaled time variable τ, the BCs have to be scaled as well.

In order to find the first two coefficients of each Fourier approximation, these terms have to be extracted from the summation and the BCs have to be applied. This leads to a system of equations which can be solved for these two coefficients. The derivation of this equation and the definition for the other two Fourier approximations can be found in Appendix B. With this new information, Equation (4.23a) can be rewritten in the following form:

$$r(\tau) = F_r + C_{a_0} a_0 + \sum_{n=3}^{n_r} \left\{ C_{a_n} a_n + C_{b_n} b_n \right\} \tag{4.29}$$

where F_r represents a constant containing all a priori information:

$$F_r = \frac{1}{2}\left(r_i - r_f\right)\cos(\pi\tau) + \frac{1}{2\pi}\left(r'_i - r'_f\right)\sin(\pi\tau) + \frac{1}{2}\left(r_i + r_f\right)\cos(2\pi\tau) + \frac{1}{4\pi}\left(r'_i + r'_f\right)\sin(2\pi\tau) \tag{4.30a}$$

while all other Fourier terms (i.e. for $n > 2$ and $n = 0$) are contained in the C_{a_n} terms, according to:

$$C_{a_o} = \frac{1}{2}\left[1 - \cos(2\pi\tau)\right] \tag{4.30b}$$

$$C_{a_n} = \begin{cases} \cos(n\pi\tau) - \cos(\pi\tau)\,; \text{ when } n \text{ is odd} \\ \cos(n\pi\tau) - \cos(2\pi\tau)\,; \text{ when } n \text{ is even} \end{cases} \tag{4.30c}$$

$$C_{b_n} = \begin{cases} \sin(n\pi\tau) - n\sin(\pi\tau)\,; \text{ when } n \text{ is odd} \\ \sin(n\pi\tau) - \frac{\pi}{2}\sin(2\pi\tau)\,; \text{ when } n \text{ is even} \end{cases} \tag{4.30d}$$

General Approach

In order to find the unknown Fourier coefficients, each equation in Equation (4.24) is again evaluated at m DPs. The number of DPs is computed with the following equation:

$$m = (N_{\text{rev}} + 1) \cdot ppr \tag{4.31}$$

where N_{rev} represents the number of revolutions of the trajectory and ppr is the corresponding points per revolution. Due to the discretisation of the equations of motion, the number of DPs should not be too low, otherwise Equation (4.27) might not be evaluated accurately. The minimum number of DPs depends both on the problem itself and on the transfer time. Unfortunately, the approved value cannot be found analytically and has to be found through trial and error.

With the definition of the discretisation points, the equations to solve for the Fourier coefficients can be set up. An example is given with the Fourier function of $r(\tau)$, but the other two variables will follow the same procedure.

$$[r]_{m\times 1} = [A_r]_{m\times(2n_r-3)}\,[X_r]_{(2n_r-3)\times 1} + [F_r]_{m\times 1} \tag{4.32}$$

in which

$$[A_r]_{m\times(2n_r-3)} = \left[C_{a_0}\ C_{a_3}\ C_{b_3}\ C_{b_4}\ \dots\ C_{a_{n_r}}\ C_{b_{n_r}}\right] \tag{4.33a}$$

$$[X_r]_{(2n_r-3)\times 1} = \left[a_0\ a_3\ b_3\ \dots\ a_{n_r}\ b_{n_r}\right]^T \tag{4.33b}$$

and $[F_r]$ is computed at each point in time according to Equation (4.30a). With this matrix equation, the components of the thrust acceleration can be expressed as a function of the states and their derivatives, such that:

$$\left[f_r\right] = f_r\left(X_r, [r], [\ddot{r}], [\dot{\theta}], [z]\right) \tag{4.34a}$$

$$\left[f_\theta\right] = f_\theta\left(X_\theta, [r], [\dot{r}], [\dot{\theta}], [\ddot{\theta}]\right) \tag{4.34b}$$

$$\left[f_z\right] = f_z\left(X_z, [r], [z], [\ddot{z}]\right) \tag{4.34c}$$

which causes the thrust acceleration to be written as a function of those three components incorporating the constraint:

$$[T_a] = \sqrt{\left[f_r\right]^2 + \left[f_\theta\right]^2 + \left[f_z\right]^2} \leq T_{a_{max}} \tag{4.35}$$

Finally, the trajectory shaping problem can be translated to a non-linear programming (NLP) problem of the following kind:

$$\begin{aligned} \min_{\mathbf{X_r}, \mathbf{X}_\theta, \mathbf{X_z}} \quad & \Delta V \\ \text{s.t.} \quad & [T_a] \leq T_{a_{max}} \end{aligned} \tag{4.36}$$

Initial Guess

To initiate the algorithm, an initial estimate for the coefficients needs to be given. Assuming n_r, n_θ and n_z are already specified, Equation (4.32) can be rewritten for X_r:

$$[X_r]_{(2n_r-3)\times 1} = \left([A_r]_{n_{App}\times(2n_r-3)}\right)^{-1}\left(\left[r_{App}\right]_{n_{App}\times 1} - [F_r]_{n_{App}\times 1}\right) \tag{4.37}$$

in which r_{App} denotes a vector r containing an approximation of the radius and n_{App} is the number of discretised data points. This quantity should be larger than m. The equation to compute n_{App} is given by:

$$n_{App} = 100 \cdot (N_{\text{rev}} + 1) \tag{4.38}$$

To find an initial estimate for r_{App} and θ_{App}, the same approach with the cubic polynomials as in Equation (4.19) is used, except for the fact that in this case they will be parametrised by the scaled time τ, instead of regular time t.

For the initial axial coordinate z however, all initial Fourier coefficients are set equal to zero. No information is available on the trend of the axial coordinate. Yet, Taheri and Abdelkhalik [22] concluded that this approach worked for all their test cases despite the fact that it might not be the best method.

II
Implementation

5 2D Implementation

This chapter will restructure the two-dimensional finite Fourier series method that was introduced in Section 4.3 in a step-by-step process to make it easier to understand how the method has been implemented in TUDAT. First a short list of actions will be given, after which each step is chronologically explained in Sections 5.1 through 5.8. The majority of the procedure is based on the work by Taheri and Abdelkhalik [21]. Where necessary, some corrections to their publication have been made and explained. Note that an overview of all these corrections can be found in Appendix D.

Algorithm Roadmap

1. Set the free parameters, i.e. departure time, the time of flight, the number of revolutions and the number of terms to be used by the Fourier series representing r and θ.

2. Compute the eight boundary conditions on the (angular) position and velocity (i.e. r_i, θ_i, r_f, θ_f, $\dot{r}_i$, $\dot{\theta}_i$, $\dot{r}_f$, $\dot{\theta}_f$).

3. With these boundary conditions, the eight coefficients of the two cubic polynomials can be computed and a linear matrix system of the form $A\mathbf{x} = B$ can be solved to obtain the initial guess.

4. The first two terms of each coefficient (i.e. a_1, a_2, b_1, b_2, c_1, c_2, d_1 and d_2) are computed by means of the boundary conditions.

5. The total travel time is discretised into m discretisation points. With this data, a matrix containing all sine and cosine terms at each discretisation point can be set up.

6. The initial guess is fed into a solver.

7. If desired, the thrust constraint can be activated as well in order to limit the maximum allowable thrust acceleration the spacecraft is allowed to experience.

8. With the output of the solver (i.e. the Fourier coefficients), the trajectory can be constructed and the ΔV can be computed.

5.1. Free Parameters

The boundary values have been set by trial and error according to Taheri and Abdelkhalik [21]. For certain bodies at certain distances, it is advised to use a certain number of terms for n_r and n_θ. The same holds for the number of revolutions, the time of flight and the number of discretisation points. However, these can be used as optimisation parameters in a later mission design stage as well. Another option is to vary them in a grid search, for example.

5.2. Boundary Conditions

The boundary conditions depend on the arrival and departure body. These values can easily be extracted from the TUDAT SPICE (Spacecraft ephemeris - Planet, satellite, comet or asteroid ephemerides and physical, dynamical and cartographic constants - Instrument information - C-matrix orientation information - Events information) interface. To reduce the computational load and improve the accuracy, the SI-units are scaled to so-called canonical units. In case of interplanetary flight, it holds that 2π time unit (TU) is equal to 1 year and that 1 distance unit (DU) is equal to 1 AU.

5.3. Initial Guess

To obtain an initial guess for the Fourier coefficients, a cubic polynomial function has been used [21]. The function has already been mentioned in Section 4.3.1, but will be stated here again for the sake of clarity:

$$r_{CP}(t) = at^3 + bt^2 + ct + d \tag{5.1a}$$

$$\theta_{CP}(t) = et^3 + ft^2 + gt + h \tag{5.1b}$$

The eight cubic polynomial coefficients can be found by inserting the eight boundary conditions from the previous step into the equations. The full derivation can be found in Appendix A. The polynomial coefficients are then defined as follows:

$$a = \frac{2\left(r_i - r_f\right) + \left(\dot{r}_i + \dot{r}_f\right) t_f}{t_f^3} \tag{5.2}$$

$$b = -\frac{3\left(r_i - r_f\right) + \left(2\dot{r}_i + \dot{r}_f\right) t_f}{t_f^2} \tag{5.3}$$

$$c = \dot{r}_i \tag{5.4}$$

$$d = r_i \tag{5.5}$$

$$e = \frac{2\left(\theta_i - \theta_f\right) + \left(\dot{\theta}_i + \dot{\theta}_f\right) t_f}{t_f^3} \tag{5.6}$$

$$f = -\frac{3\left(\theta_i - \theta_f\right) + \left(2\dot{\theta}_i + \dot{\theta}_f\right) t_f}{t_f^2} \tag{5.7}$$

$$g = \dot{\theta}_i \tag{5.8}$$

$$h = \theta_i \tag{5.9}$$

With the cubic polynomial coefficients and the number of Fourier terms, the initial guess can be computed. In this description only the case for the radius r is explained, as the transfer angle θ follows the exact same procedure. The only difference is that the n_r term needs to be replaced by n_θ. The total time of flight is discretised into $2n_r + 1$ points at which the polynomials in Equation (5.1) are evaluated. The same holds for the Fourier series in Equation (4.7a). This results in a matrix system of the form $A\mathbf{x} = B$. All Fourier terms are stored in A and thus it becomes a square $(2n_r + 1)$ x $(2n_r + 1)$ matrix. The coefficients are stored in $\mathbf{x}$, which is a $(2n_r + 1)$ x 1 vector. Finally, the B vector contains all outcomes of the cubic polynomials, which means it is a $(2n_r + 1)$ x 1 vector as well. In this linear system, the solution of $\mathbf{x}$ can easily be found by left-multiplying B with A^{-1}. The introduced matrix and vectors contain the following terms:

$$A = \begin{bmatrix} \frac{1}{2} & \cos\left(\frac{\pi}{T}t_0\right) & \cos\left(\frac{2\pi}{T}t_0\right) & \cdots & \cos\left(\frac{n_r\pi}{T}t_0\right) & \sin\left(\frac{\pi}{T}t_0\right) & \sin\left(\frac{2\pi}{T}t_0\right) & \cdots & \sin\left(\frac{n_r\pi}{T}t_0\right) \\ \frac{1}{2} & \cos\left(\frac{\pi}{T}t_1\right) & \cos\left(\frac{2\pi}{T}t_1\right) & \cdots & \cos\left(\frac{n_r\pi}{T}t_1\right) & \sin\left(\frac{\pi}{T}t_1\right) & \sin\left(\frac{2\pi}{T}t_1\right) & \cdots & \sin\left(\frac{n_r\pi}{T}t_1\right) \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ \frac{1}{2} & \cos\left(\frac{\pi}{T}t_i\right) & \cos\left(\frac{2\pi}{T}t_i\right) & \cdots & \cos\left(\frac{n_r\pi}{T}t_i\right) & \sin\left(\frac{\pi}{T}t_i\right) & \sin\left(\frac{2\pi}{T}t_i\right) & \cdots & \sin\left(\frac{n_r\pi}{T}t_i\right) \end{bmatrix} \tag{5.10}$$

$$\mathbf{x} = \begin{bmatrix} a_0 & a_1 & a_2 & \cdots & a_{n_r} & b_1 & b_2 & \cdots & b_{n_r} \end{bmatrix}^T \tag{5.11}$$

$$B = \begin{bmatrix} r_{CP}(t_0) & r_{CP}(t_1) & \cdots & r_{CP}(t_i) \end{bmatrix}^T \tag{5.12}$$

5.4. Initial Coefficients

The Fourier series in Equation (4.7) and their derivatives can be combined with the eight boundary conditions to form a system of equations from which the first eight coefficients of each Fourier term can be obtained as a function of the other (higher-order) terms. This is done because the first terms are more dominant than the higher-order terms, hence they will influence the trajectory shape more significantly [22]. An exact derivation of the systems of equations and their solution can be found in Appendix B. Here only the final solutions are stated:

$$a_1 = \frac{r_i - r_f}{2} - \sum_{n=3}^{n_r} a_n;\ n_r \geq 3,\ n:\text{odd} \tag{5.13a}$$

$$a_2 = \frac{r_i + r_f - a_0}{2} - \sum_{n=4}^{n_r} a_n;\ n_r \geq 4,\ n:\text{even} \tag{5.13b}$$

$$b_1 = \frac{T}{2\pi}\left(\dot{r}_i - \dot{r}_f\right) - \sum_{n=3}^{n_r} nb_n;\ n_r \geq 3,\ n:\text{odd} \tag{5.13c}$$

$$b_2 = \frac{T}{4\pi}\left(\dot{r}_i + \dot{r}_f\right) - \frac{1}{2}\sum_{n=4}^{n_r} nb_n;\ n_r \geq 4,\ n:\text{even} \tag{5.13d}$$

$$c_1 = \frac{\theta_i - \theta_f}{2} - \sum_{n=3}^{n_\theta} c_n;\ n_\theta \geq 3,\ n:\text{odd} \tag{5.13e}$$

$$c_2 = \frac{\theta_i + \theta_f - c_0}{2} - \sum_{n=4}^{n_\theta} c_n;\ n_\theta \geq 4,\ n:\text{even} \tag{5.13f}$$

$$d_1 = \frac{T}{2\pi}\left(\dot{\theta}_i - \dot{\theta}_f\right) - \sum_{n=3}^{n_\theta} nd_n;\ n_\theta \geq 3,\ n:\text{odd} \tag{5.13g}$$

$$d_2 = \frac{T}{4\pi}\left(\dot{\theta}_i + \dot{\theta}_f\right) - \frac{1}{2}\sum_{n=4}^{n_\theta} nd_n;\ n_\theta \geq 4,\ n:\text{even} \tag{5.13h}$$

5.5. Sine and Cosine Matrices

The final data the solver requires are two matrices with all sine and cosine terms for both Fourier series. By discretising the time of flight by m discretisation points and storing all trigonometric terms, much time is saved during the solving procedure, as the terms only need to be retrieved, instead of being evaluated at every iteration. The matrices for the radius r and transfer angle θ are m x $2n_r$ and m x $2n_\theta$, respectively. Matrix R, containing all terms for the Fourier series describing r, is expressed as follows:

$$R = \begin{bmatrix} \cos\left(\frac{\pi t_0}{T}\right) & \cos\left(\frac{2\pi t_0}{T}\right) & \cdots & \cos\left(\frac{n_r \pi t_0}{T}\right) & \sin\left(\frac{\pi t_0}{T}\right) & \sin\left(\frac{2\pi t_0}{T}\right) & \cdots & \sin\left(\frac{n_r \pi t_0}{T}\right) \\ \cos\left(\frac{\pi t_1}{T}\right) & \cos\left(\frac{2\pi t_1}{T}\right) & \cdots & \cos\left(\frac{n_r \pi t_1}{T}\right) & \sin\left(\frac{\pi t_1}{T}\right) & \sin\left(\frac{2\pi t_1}{T}\right) & \cdots & \sin\left(\frac{n_r \pi t_1}{T}\right) \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ \cos\left(\frac{\pi t_{m-1}}{T}\right) & \cos\left(\frac{2\pi t_{m-1}}{T}\right) & \cdots & \cos\left(\frac{n_r \pi t_{m-1}}{T}\right) & \sin\left(\frac{\pi t_{m-1}}{T}\right) & \sin\left(\frac{2\pi t_{m-1}}{T}\right) & \cdots & \sin\left(\frac{n_r \pi t_{m-1}}{T}\right) \end{bmatrix} \tag{5.14}$$

5.6. Solving Procedure

To eventually solve for the unknown Fourier coefficients, a non-linear solver is required, as it contains quadratic terms and products of terms (recall from Equation (4.13)):

$$f\left(r, \dot{r}, \ddot{r}, \dot{\theta}, \ddot{\theta}\right) = r^2\left(\dot{\theta}\ddot{r} - \dot{r}\ddot{\theta}\right) + \dot{\theta}\left(\mu - 2r\dot{r}^2\right) - \left(r\dot{\theta}\right)^3 = 0 \tag{5.15}$$

Due to its previously-proven compliance with shape-based methods by Vroom [25] and Gondelach [6], the Nelder-Mead method has been chosen. More on this solver can be read in Section 7.1. The solver evaluates Equation (5.15) at each DP and tries to minimise the sum of its squared residuals according to:

$$\min\left(\sum_{n=1}^{m} f\left(a_0, a_1, \cdots, a_{n_r}, b_1, \cdots, b_{n_r}, c_0, c_1, \cdots, c_{n_\theta}, d_1, \cdots, d_{n_\theta}; t_n\right)\right)^2 \tag{5.16}$$

The solver requires all above-mentioned information, such as the free parameters, the time discretisation vector, the initial eight coefficients and the sine and cosine matrices. In case it keeps getting trapped in a local minimum that provides an infeasible trajectory, some randomisation is introduced to the initial guess by means of a normal distribution that gets its average value and variance from the content of the initial guess vector. Furthermore, the size of the decision vector **x** is important. The correct implementation requires the following vector to be fed to the solver:

$$\mathbf{x} = \begin{bmatrix} a_0 & a_3 & a_4 & \cdots & a_{n_r} & b_3 & b_4 & \cdots & b_{n_r} & c_0 & c_3 & c_4 & \cdots & c_{n_\theta} & d_3 & d_4 & \cdots & d_{n_\theta} \end{bmatrix} \tag{5.17}$$

This decision vector has a size of ($2n_r$ + $2n_\theta$ - 6), which is exactly equal to the total number of Fourier terms minus the first two coefficients of each sine and cosine term that are determined in Equation (5.13a) through (5.13h). Note that the values that are fed into the solver only consist of the first part of these equations, i.e. the summation terms are excluded, as these are added in the solving process by retrieving them from the decision vector. Due to the ambiguity of the right method of the algorithm as described by Taheri and Abdelkhalik [21], first a slightly different method had been implemented. Further details and an explanation why this method cannot not be correct is found in Section 8.1 and Appendix D.

5.7. Thrust Constraint

If desired, the thrust constraint can be activated. This great asset, which is inherent to the finite Fouries series method, allows for a maximum thrust acceleration value to be set. The constraint is formulated as follows:

$$C = \left(\frac{T_a}{T_{a_{max}}}\right)^2 \leq 1 \tag{5.18}$$

in which the thrust constraint C is the square of the ratio of the computed thrust acceleration T_a and the maximum thrust acceleration $T_{a_{max}}$. This value needs to be smaller than or equal to one.

5.8. Trajectory Construction and Velocity Increment

When all coefficients are known the trajectory shape can be constructed from the spatial coordinates, while the total ΔV can be determined after the thrust profile has been constructed. These terms can then be numerically integrated in order to get the required ΔV.

6
3D Implementation

Whereas Chapter 5 contained a step-by-step guide of implementation of the two-dimensional finite Fourier series method, this chapter will elaborate on the three-dimensional implementation. First a short list of actions will be given, after which each step is chronologically explained in Sections 6.1 through 6.7. The majority of the procedure is based on the work by Taheri and Abdelkhalik [22]. In addition to the two-dimensional corrections, changes that have been made to the three-dimensional version are summarised in Appendix D as well.

Algorithm Roadmap

1. Set the free parameters, i.e. departure time, the time of flight, the number of revolutions and the terms to be used by the Fourier series representing r, θ and z.

2. Compute the twelve boundary conditions on position and velocity. (i.e. r_i, θ_i, z_i, r_f, θ_f, z_f, $\dot{r}_i$, $\dot{\theta}_i$, $\dot{z}_i$ $\dot{r}_f$, $\dot{\theta}_f$, $\dot{z}_f$).

3. Having converted them to the scaled time domain, the eight coefficients of the two cubic polynomials can be computed and a linear matrix system of the form $A\mathbf{x} + C = B$ can be solved to obtain the initial guess.

4. The first two terms of each coefficient (i.e. a_1, a_2, b_1, b_2, c_1, c_2, d_1, d_2, e_1, e_2, f_1 and f_2) are computed by means of the boundary conditions after which they are stored in a separate vector.

5. The total travel time is discretised into m discretisation points. With this data, a matrix containing all sine and cosine terms at those discretisation points can be set up.

6. The initial guess is fed into a solver.

7. With the output of the solver (i.e. the Fourier coefficients), the trajectory can be constructed and the ΔV can be computed.

6.1. Free Parameters

The boundary values largely depend on the problem at hand. If the finite Fourier series is used to do a grid search, all mentioned parameters will be varied over a certain region, except for the number of Fourier terms, which is usually fixed. For certain bodies at certain distances, it is advised to use a certain number of terms for n_r, n_θ and n_z, however, these can be used as optimisation parameters in a later mission design stage as well.

6.2. Boundary Conditions

The boundary conditions depend on the arrival and departure body. These values can easily be extracted from the TUDAT SPICE interface. Also, as mentioned in Section 5.2, a scaling to so-called canonical units has been applied to reduce the computational load and improve the accuracy.

6.3. Initial Guess

To obtain an initial guess for the Fourier coefficients, a cubic polynomial function has been used [22]. The function has already been mentioned in Section 4.3.1, but note that this version uses the scaled time τ instead of the standard time t:

$$r_{CP}(\tau) = a\tau^3 + b\tau^2 + c\tau + d \tag{6.1a}$$

$$\theta_{CP}(\tau) = e\tau^3 + f\tau^2 + g\tau + h \tag{6.1b}$$

The eight cubic polynomial coefficients can be found by inserting the first eight boundary conditions from the previous step into the equations. This will lead to a system of eight equations, which can be solved linearly. The full derivation is nearly identical to the one in Chapter 5, which can be found in Appendix A. The polynomial coefficients are then defined as according to:

$$a = r'_f + r'_i + 2\left(r_i - r_f\right) \tag{6.2}$$

$$b = 3\left(r_f - r_i\right) - 2r'_i - r'_f \tag{6.3}$$

$$c = r'_i \tag{6.4}$$

$$d = r_i \tag{6.5}$$

$$e = \theta'_f + \theta'_i + 2\left(\theta_i - \theta_f\right) \tag{6.6}$$

$$f = 3\left(\theta_f - \theta_i\right) - 2\theta'_i - \theta'_f \tag{6.7}$$

$$g = \theta'_i \tag{6.8}$$

$$h = \theta_i \tag{6.9}$$

6.4. Initial Coefficients

The Fourier series in Equation (4.23) and their derivatives can be combined with the 12 boundary conditions to form a system of equations from which the first 12 coefficients of each Fourier term can be isolated. This is done because the first terms are more dominant regarding the convergence of the solver than higher-order terms, hence they will influence the trajectory shape more significantly [21]. This will also narrow the solution space and improve the convergence speed of the solver. An exact derivation of the systems of equations and their solution can be found in Appendix C. Here only the final solution of the vector containing all boundary conditions is stated:

$$F_r = \frac{1}{2}\left(r_i - r_f\right)\cos(\pi\tau) + \frac{1}{2\pi}\left(r'_i - r'_f\right)\sin(\pi\tau) + \frac{1}{2}\left(r_i + r_f\right)\cos(2\pi\tau) + \frac{1}{4\pi}\left(r'_i + r'_f\right)\sin(2\pi\tau) \tag{6.10}$$

As can be seen from Equation (6.10), the expression is a function of the scaled time τ. This entails that its value changes depending on the DP it is evaluated at, resulting in m different outputs. These can be assembled in a column vector of size (m x 1):

$$F_r = \begin{bmatrix} F_r(\tau_0) & F_r(\tau_1) & \cdots & F_r(\tau_m) \end{bmatrix}^T \tag{6.11}$$

The exact same method is applied to the other two dimensions (i.e. θ and z). In Appendix C the full representation of the different F equations and matrices can be found.

6.5. Sine and Cosine Matrices

To speed up the solving process, all terms containing a sine or cosine component have been stored in matrices before the solving process is initiated. These matrices are pushed into the solver as well. In this way the algorithm only needs to fetch the right trigonometric term instead of computing it at each iteration. The matrices are set up by discretising the time of flight by m discretisation points. The matrices for the radius r, transfer angle θ and the axial component z have the following sizes: $(m \text{ x } 2n_r - 3)$, $(m \text{ x } 2n_\theta - 3)$ and $(m \text{ x } 2n_z - 3)$, respectively. Matrix A_r, containing all terms for the Fourier series describing r is expressed as follows:

$$A_r = \begin{bmatrix} C_{a_0}(\tau_0) & C_{a_3}(\tau_0) & C_{b_3}(\tau_0) & C_{a_4}(\tau_0) & C_{b_4}(\tau_0) & \cdots & C_{a_{n_r}}(\tau_0) & C_{b_{n_r}}(\tau_0) \\ C_{a_0}(\tau_1) & C_{a_3}(\tau_1) & C_{b_3}(\tau_1) & C_{a_4}(\tau_1) & C_{b_4}(\tau_1) & \cdots & C_{a_{n_r}}(\tau_1) & C_{b_{n_r}}(\tau_1) \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \cdots & \vdots \\ C_{a_0}(\tau_m) & C_{a_3}(\tau_m) & C_{b_3}(\tau_m) & C_{a_4}(\tau_m) & C_{b_4}(\tau_m) & \cdots & C_{a_{n_r}}(\tau_m) & C_{b_{n_r}}(\tau_m) \end{bmatrix} \tag{6.12}$$

in which each C_{a_n} or C_{b_n} term is represented by:

$$C_{a_n} = \begin{cases} \cos(n\pi\tau) - \cos(\pi\tau); & n \text{ is odd} \\ \cos(n\pi\tau) - \cos(2\pi\tau); & n \text{ is even} \end{cases} \tag{6.13a}$$

$$C_{b_n} = \begin{cases} \sin(n\pi\tau) - n\sin(\pi\tau); & n \text{ is odd} \\ \sin(n\pi\tau) - \frac{n}{2}\sin(2\pi\tau); & n \text{ is even} \end{cases} \tag{6.13b}$$

In a similar way, these matrices can also be constructed for the other two dimensions and their derivatives. An overview hereof is found in Appendix C.

6.6. Solving Procedure

Eventually, the problem can be described by the following equation:

$$\begin{aligned} \min_{\mathbf{X_r},\mathbf{X_\theta},\mathbf{X_z}} \quad & \Delta V \\ \text{s.t.} \quad & [T_a] \leq T_{a_{max}} \end{aligned} \tag{6.14}$$

which says that the ΔV needs to be minimised such that the thrust acceleration T_a is less than the maximum value $T_{a_{max}}$. This last option is however optional, but the core of the problem is that the ΔV is a function of the decision vector containing the Fourier coefficients in all three dimensions according to:

$$\mathbf{x} = \begin{bmatrix} \mathbf{X_r} & \mathbf{X_\theta} & \mathbf{X_z} \end{bmatrix}^T \tag{6.15}$$

in which the three separate terms are described by:

$$\mathbf{X_r} = \begin{bmatrix} a_0 & a_3 & b_3 & \cdots & a_{n_r} & b_{n_r} \end{bmatrix}^T \tag{6.16a}$$

$$\mathbf{X_\theta} = \begin{bmatrix} c_0 & c_3 & d_3 & \cdots & c_{n_\theta} & d_{n_\theta} \end{bmatrix}^T \tag{6.16b}$$

$$\mathbf{X_z} = \begin{bmatrix} e_0 & e_3 & f_3 & \cdots & e_{n_z} & f_{n_z} \end{bmatrix}^T \tag{6.16c}$$

The total thrust acceleration at each moment in time is found by adding the three separate components, which are expressed by:

$$\begin{cases} \ddot{r} - r\dot{\theta}^2 + \frac{\mu}{s^3} r = f_r \\ 2\dot{r}\dot{\theta} + r\ddot{\theta} = f_\theta \\ \ddot{z} + \frac{\mu}{s^3} z = f_z \end{cases} \tag{6.17}$$

The root of the squared sum of these equations is equal to the thrust acceleration as depicted in Equation (6.18). Even though Equation (6.14) states that the ΔV is minimised, it is effectively the T_a that is minimised, as the ΔV is obtained from the thrust acceleration by means of integration over time. This means that the ΔV scales directly proportionally with the thrust acceleration.

$$T_a = \sqrt{f_r^2 + f_\theta^2 + f_z^2} \tag{6.18}$$

6.7. Trajectory Construction and Velocity Increment

When the solver has converged, the trajectory shape and the thrust profile can be constructed from the newly obtained Fourier coefficients. This then also determines the required ΔV, as it is found by numerically integrating this thrust profile.

7 Validation

Before a model can actually be used for further research, it needs to be validated properly. This chapter contains everything concerning the trustworthiness of the implementation of the method. In Section 7.1 the choice for the Nelder-Mead algorithm is argumented, after which, in Section 7.2, the two-dimensional finite Fourier series is validated against one case study to Mars. The three-dimensional extension is then validated against two cases. In Section 7.3 another trajectory study to Mars is done, while Section 7.4 describes the case study to the comet Tempel-1.

7.1. Solver Validation

To ensure that the chosen solver is the right one for this particular problem, it has been tested with an optimisation test function to assess its performance. In Section 7.1.1 the solving algorithm is explained after which it is tested against the Schwefel function in Section 7.1.2.

7.1.1. Nelder-Mead Simplex Method

Also referred to as the downhill simplex or the amoeba algorithm, the Nelder-Mead simplex method is a local optimisation approach for searching the space of n-dimensional vectors. The method uses $n+1$ points to generate a polytope in an n-dimensional space. This entails that it looks like a line in one dimension, a triangle in two dimensions and a tetrahedron in three dimensions. These shapes are called simplices.

Once the method has been initialised, it changes the simplex step by step, mostly by moving the point of the simplex with the largest function value through the opposite face of the simplex to a point with a lower value. This starting point is called $\mathbf{P}_0$, which allows for the other n points to be defined as:

$$\mathbf{P}_i = \mathbf{P}_0 + \Delta \mathbf{e}_i \tag{7.1}$$

in which $\mathbf{e}_i$ represents the i^{th} unit vector and where Δ is a constant that denotes the step size. The latter one can be specified such that it is different for each of the directions the unit vectors are pointing in [15]. The steps the algorithm will be taking are called reflections. They make sure the simplex stays nondegenerate, i.e. ensuring the surface encloses a finite n-dimensional inner surface. An overview of the sundry reflections can be found in Figure 7.1. Once the simplex reaches a valley floor, it moves down into the valley, and finally the simplex will contract around the optimum. The Nelder-Mead method has converged when the simplex changes shape only marginally within the set tolerances [29].

7.1.2. Schwefel Test Function

To validate the implementation of the randomisation of the initial guess vector in combination with the Nelder-Mead algorithm, they have been applied to the Schwefel function. This entails that the initial guess is improved by means of the Nelder-Mead algorithm and if it gets stuck in a local minimum, a random push is given to steer it into the right direction. The Schwefel function is a widely used optimisation test function, as it has many local minima that are close to each other. The function, in which d denotes the dimension of the problem, is defined as follows in Equation (7.2):

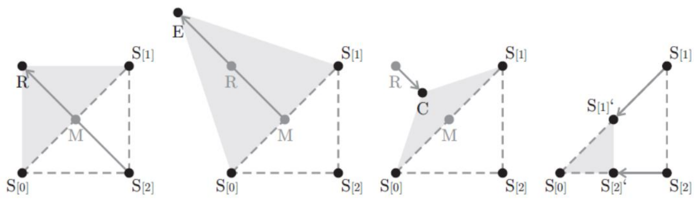


Figure 7.1: The possible steps of the simplices used in the Nelder-Mead simplex method. From left to right the following is shown: a reflection, an expansion, a contraction and a shrinking [29].

$$f(\mathbf{x}) = 418.9829d - \sum_{i=1}^{d} x_i \sin\left(\sqrt{|x_i|}\right) \tag{7.2}$$

To test the algorithm, two dimensions have been used. The input domain for the function is the hypercube defined by $x_i \in$ [-500,500] for all i = 1, ..., d. With this data, Figure 7.2 can be generated [2].

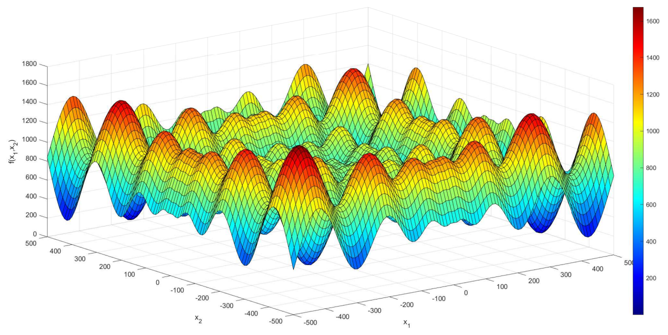


Figure 7.2: The Schwefel function on the standard domain x_1, $x_2 \in$ [-500,500].

It turned out that the algorithm required 133 pushes to finally end up at the value of $f(\mathbf{x}) = 2.55 \times 10^{-5}$ at $\mathbf{x}$ = [420.9687,420.9687], which is the global minimum [2]. This has proven that the algorithm is capable of finding the global minimum with a high degree of accuracy, hence validating its use. Important input variables are shown in Table 7.1.

Table 7.1: The settings that have been used for the validation of the randomisation and Nelder-Mead algorithm.

Criterion	Value
Maximum number of iterations	1000
Nelder-Mead threshold	1×10^{-15}
Solution treshold	1×10^{-3}
Distribution seed	43.0
Distribution	Uniform

7.2. Mars in 2D

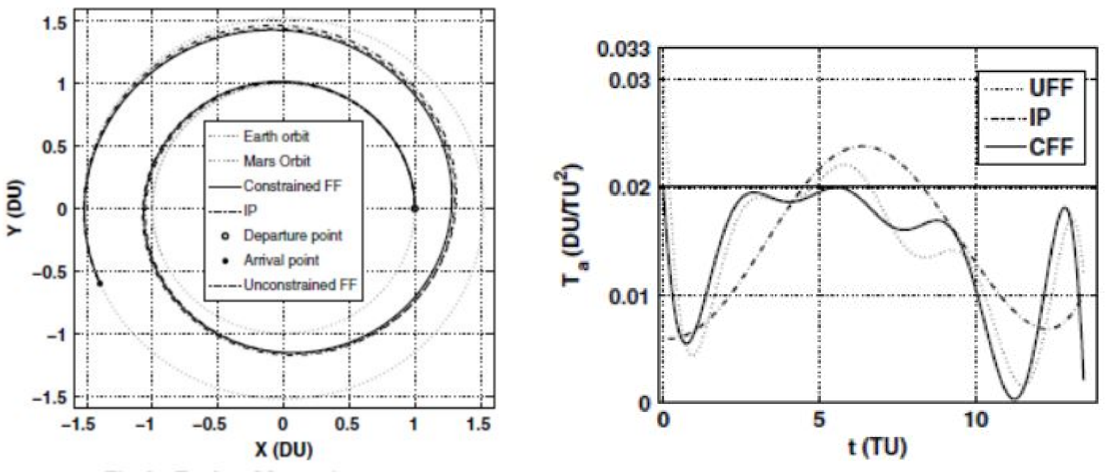
In Table 7.2 the various boundary conditions and input parameters to compute a trajectory from Earth to Mars can be found [21]. These values are used as a reference value to test whether the method is functioning properly. In Figures 7.3a and 7.3b the exact trajectory and the corresponding thrust profile (in canonical units) are shown.

Recall that the canonical units that are used on an interplanetary scale are TU and DU, which denote a time unit and a distance unit respectively. One distance unit corresponds to one astronomical unit, while the time unit is defined such that 2π TU equals one year. This scaling prevents the programme from using large numbers.

Finally, a claim that at least 15 DPs are needed to fully capture the trajectory topologies and that more than 80 DPs does not yield more accurate results, combined with the mere graphical results, are all validation means Taheri and Abdelkhalik [21] provided. Note that the abbreviations UFF and CFF represent the unconstrained finite Fourier series and the constrained finite Fourier series, respectively.

Table 7.2: Input parameters and boundary conditions for the trajectory from Earth to Mars [21].

Boundary Conditions		Input Parameters	
r_i	1 DU	N_{rev}	1
θ_i	0 rad	n_r	2
r_f	1.5234 DU	n_θ	5
θ_f	9.831 rad	$T_{a_{max}}$	0.02 DU/TU^2
$\dot{r}_i$	0 DU/TU	# DP	22
$\dot{\theta}_i$	1 rad/TU	TOF	13.447 TU
$\dot{r}_f$	0 DU/TU		
$\dot{\theta}_f$	0.5318 rad/TU		



(a) The trajectory from Earth to Mars.

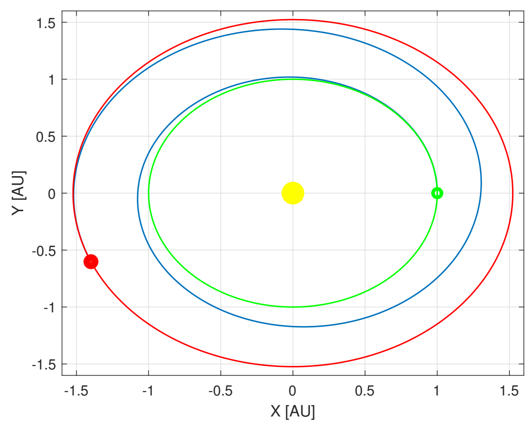
(b) The thrust profile for the trajectory from Earth to Mars.

Figure 7.3: The trajectory from Earth to Mars and the corresponding thrust profile [21].

The computed trajectories by both the unconstrained and the constrained Fourier series can be observed in Figures 7.4a and 7.4b. It takes an eagle's eye to see any difference between the two transfer orbits, but it appears to be in line with Figure 7.3a.

A more serious distinction between the two methods can be noticed when looking at the thrust profiles in Figure 7.5. The red line resembles the thrust constraint of 0.02 TU/DU^2, which corresponds to 0.11 mm/s^2. In Figure 7.5a the thrust constraint is not activated and thus the trajectory is not limited in any way. Approximately halfway it passes the constraint line. On the other hand, the constrained trajectory in Figure 7.5b perfectly obeys the thrust acceleration limit and only shortly touches the line. Comparing the results to Figure 7.3b, both the shape and the order of magnitude are in line with Taheri and Abdelkhalik [21].

Furthermore, the claim that at least 15 DPs are required to accurately capture the trajectory and that working with more than 80 DPs does not enhance the accuracy, is examined. In Figure 7.6 the required ΔV for the transfer orbit is plotted against the number of DPs that have been used to correctly capture that orbit. It clearly shows the asymptotic behaviour that is expected when more DPs are used. Note that the graph starts at 15 DPs and ends at 500 DPs. From the plot it can be concluded that indeed the increase in accuracy beyond 80 DPs is negligible.


(a) The trajectory obtained by the unconstrained finite Fourier series.

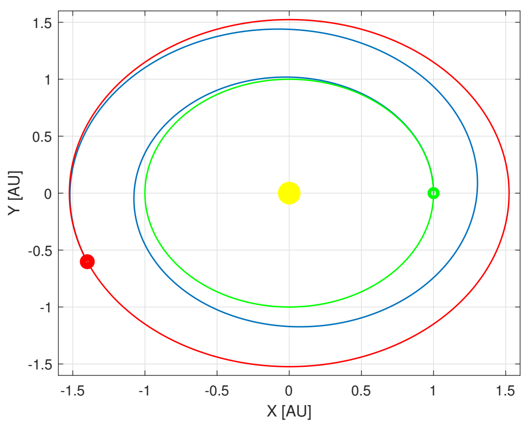
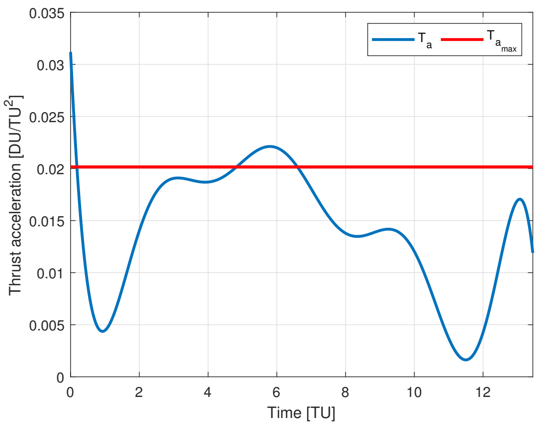

(b) The trajectory obtained by the constrained finite Fourier series.

Figure 7.4: The unconstrained (left) and constraint (right) trajectories.


(a) The thrust profile corresponding to the unconstrained trajectory.

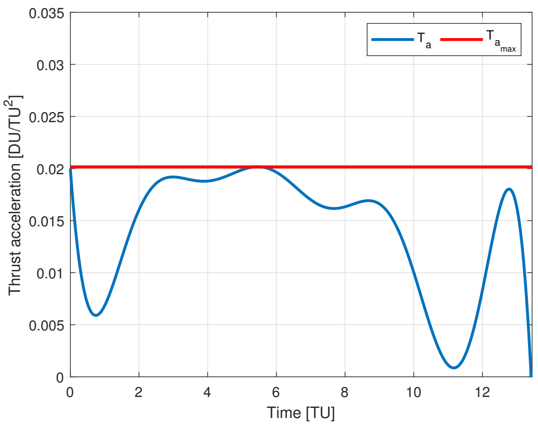

(b) The thrust profile corresponding to the constrained trajectory.

Figure 7.5: The unconstrained (left) and constrained (right) thrust profile.

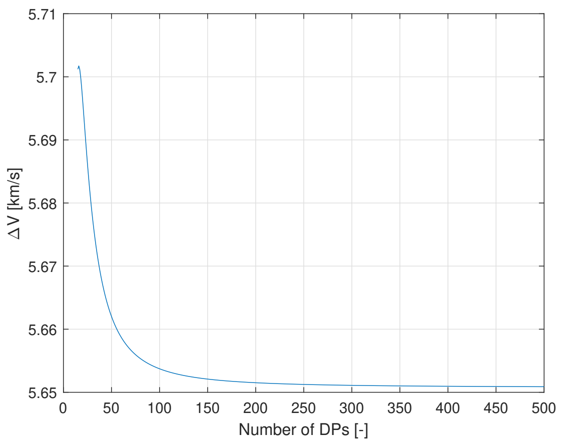
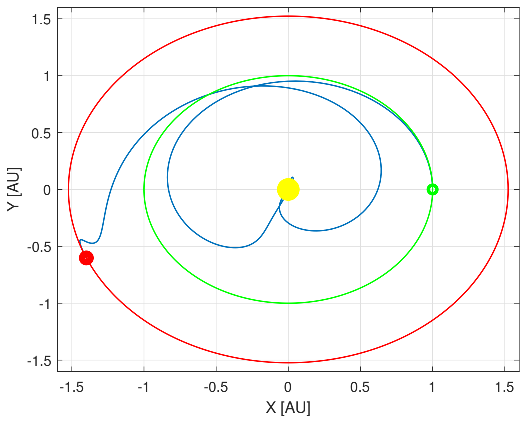
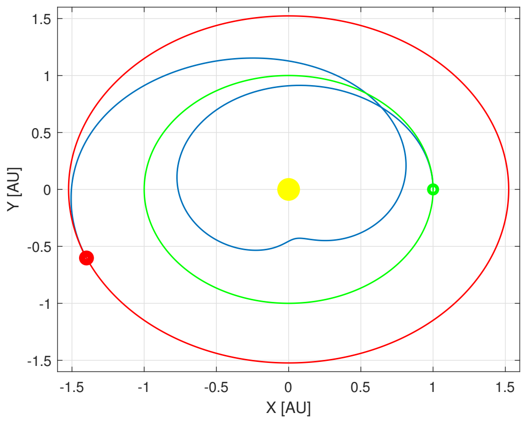


Figure 7.6: The ΔV shows asymptotic behaviour when more discretisation points are used, conforming Taheri and Abdelkhalik [21].

Finally, in Figures 7.7a and 7.7b clear evidence is presented that the finite Fourier series is not capable of capturing trajectories correctly when less than 15 DPs are used. The two examples show a transfer orbit that has been generated with the data from Table 7.2, but this time only eight and twelve DPs have been used.



(a) A transfer orbit to Mars when eight DPs are used.



(b) A transfer orbit to Mars when twelve DPs are used.

Figure 7.7: Two examples of transfer orbits that have been found with an insufficient number of DPs with the unconstrained finite Fourier series.

7.3. Mars in 3D

Mars has proven to be a popular target in the field of trajectory design and therefore it has also been used as a reference case for the three-dimensional finite Fourier series method, just as for the hodographic shaping method and the spherical shaping method [5, 12, 22]. In Figures 7.8 to 7.11 the outcome of the comparison of the results from Taheri and Abdelkhalik [22] and the author will be given in the form of a porkchop plot, the three-dimensional trajectory and the corresponding thrust profile. The reference values to set up this validation case can be found in Table 7.3.

Table 7.3: The input parameters that have been used to generate the trajectories from Earth to Mars. † The true value of this is disputable, as in Taheri [20] a *ppr* value of 10 is mentioned, while Taheri and Abdelkhalik [22] use 30 *ppr*.

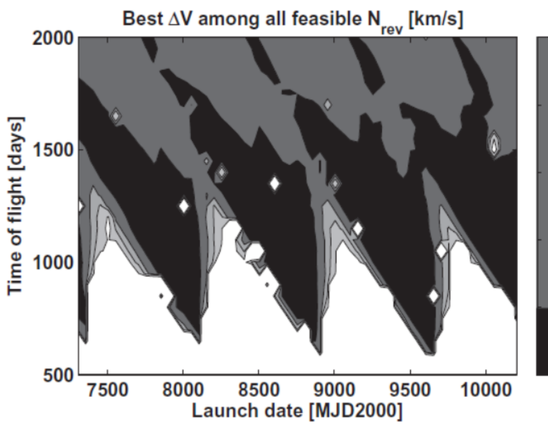
Input parameter	Unit
$T_{a_{max}}$ [m/s^2]	1.5×10^{-4}
ΔT Departure date [days]	50
ΔT Time of flight [days]	50
Launch date range [MJD2000]	7304.5-10225.5
Time of flight bounds [days]	500-2000
Points per revolution [-]	30†
N_{rev} range [-]	1-4
n_r [-]	6
n_θ [-]	6
n_z [-]	4

The porkchop plots that have been generated with the data from Table 7.3 can be found in Figure 7.8. Note that the white space in the reference plot in Figure 7.8a indicates that the trajectory is infeasible. This is a criterion that has not been specified any further by Taheri and Abdelkhalik [22], hence it has been decided to actually include all computed solutions in Figure 7.8b. This required the scale to be set properly, as the actual ΔV ranges up to values of 154 km/s. Therefore it is important to note that the white arrow on top of the colour-bar indicates that the scaling is cut of at 10 km/s, which entails that the white areas contain a large variety of possibilities that are deemed infeasible by Taheri and Abdelkhalik [22]. Overall, Figures 7.8a and 7.8b show considerable similarities. The overall shape of the porkchop plot, which is caused by the periodic motion of the Earth and Mars, can be clearly seen, for example.

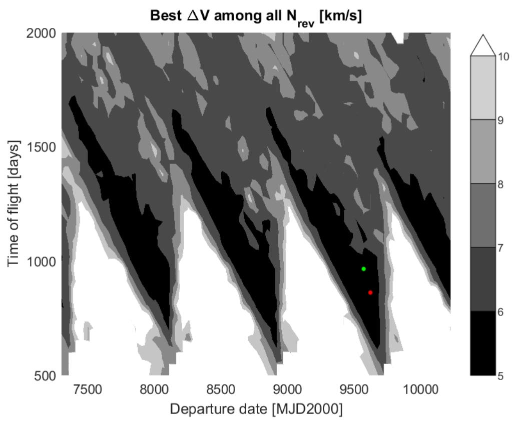
The ΔV regions however seem to differ somewhat. This phenomenon can be attributed to the occurrence of infeasible regions Taheri and Abdelkhalik [22] did include, whereas Figure 7.8b contains all computed trajectory solutions.

Furthermore, the quality of the plot by Taheri and Abdelkhalik [22] is controversial, as their scaling only shows two colours for the entire ΔV regime, while Figure 7.8a clearly shows 4 different colours. This is especially visible at the borders of the white triangular regions at the bottom.

Moreover, the departure date and the time of flight of the trajectory that requires the least amount of ΔV slightly differ. In Figure 7.8b the difference is indicated by a green and a red dot. The green dot resembles the optimal trajectory that has been found by Taheri and Abdelkhalik [22], while the red dot depicts the optimal trajectory the author has found.



(a) The reference porkchop plot by Taheri and Abdelkhalik [22].



(b) The computed porkchop plot.

Figure 7.8: A comparison between the best porkchop plots computed by Taheri and Abdelkhalik [22] (left) and the author (right).

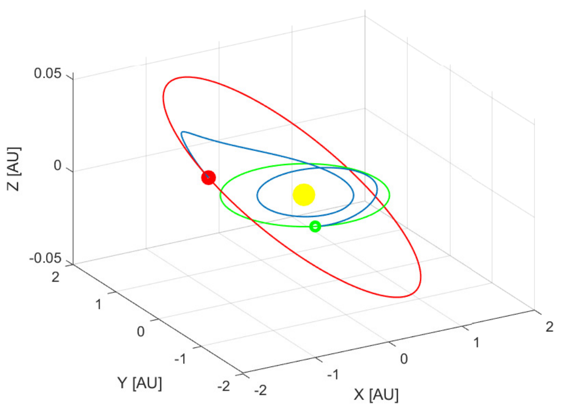
To clarify the definition of the so-called infeasible regions in Figure 7.8 a few examples have been presented in Figure 7.9. It shows two different trajectories and their corresponding thrust profiles. An infeasible trajectory is characterised by the following two properties: either their thrust profile or their trajectory is unrealistic.

In Figure 7.9a a nice example of the first is shown. The trajectory looks perfectly normal, but with a ΔV of 28.6 km/s this is definitely not the optimal solution and when taking a closer look at the thrust profile in Figure 7.9c it is evident that the thrust constraint is exceeded severely, hence the trajectory cannot be flown.

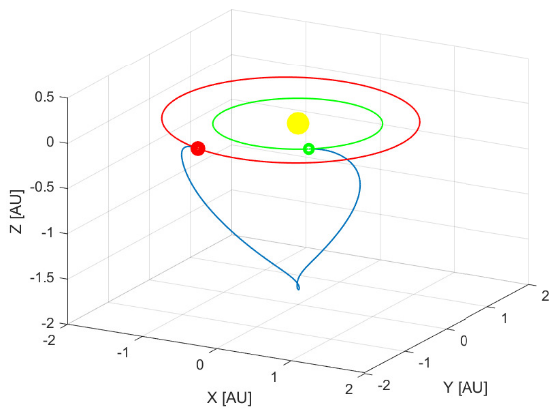
Additionally, whilst analysing Figure 7.9b one quickly comes to the conclusion that the trajectory itself is impossible to fly. Especially because the spacecraft would have to descent nearly 2 AU after which it makes a hairpin bend in order to ascend towards the final destination again. With such a trajectory it is no surprise that the thrust profile in Figure 7.9d never complies to the imposed constraint of 15 mm/s^2. This makes it a clear example of an infeasible trajectory.

If then the optimal solution is analysed from a perspective of minimal ΔV and compared to the solution found in literature, there is a visible discrepancy. The two trajectories are depicted in Figure 7.10. The trajectory that was obtained by Taheri and Abdelkhalik [22] in Figure 7.10a seems to end at a higher position in the positive z-direction. This difference can be explained by looking at the grid that has been used for the porkchop plots. Both the time of flight and the departure date are entered with an interval of 50 days. The optimal result the author has found requires a time of flight which is 100 days shorter, while the departure date is about 50 days later, hence the absolute arrival time will 50 days earlier as well.

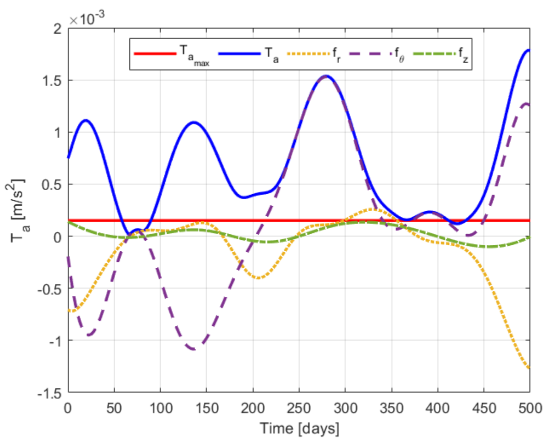
Furthermore, looking at the underlying thrust profile of these trajectories in Figure 7.11, there is not much that stands out. The overall shape of the thrust components is largely the same: It starts with the maximum allowed thrust acceleration value, after which it drops to a minimum after about 400 days. From this point onwards the reference trajectory in Figure 7.11a assumes the shape of a lumpy concave parabola, while Figure 7.11b shows two steep increases.



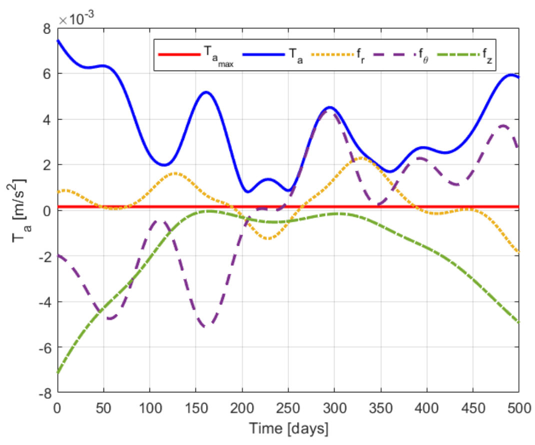
(a) The departure MJD2000 is 7453 and the time of flight is 500 days.



(b) The departure MJD2000 is 8245 and the time of flight is 500 days.

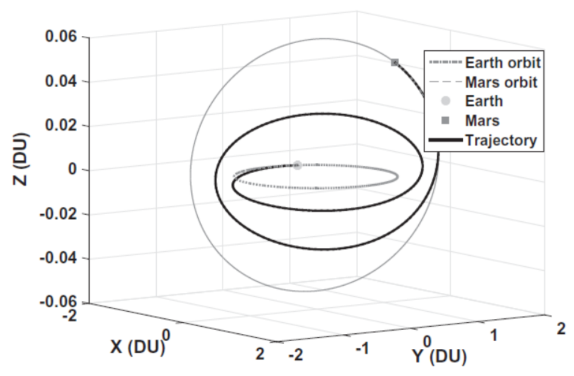


(c) The thrust profile corresponding to Figure 7.9a.

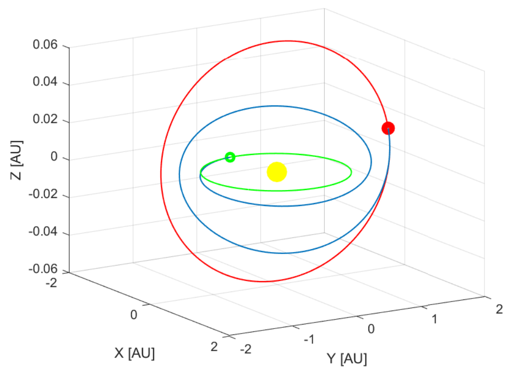


(d) The computed thrust profile corresponding to Figure 7.9b.

Figure 7.9: An example of two trajectory solutions and their thrust profiles that have been deemed infeasible.



(a) The reference trajectory [22]. The departure MJD2000 is 9554.5 and the time of flight is 950 days.

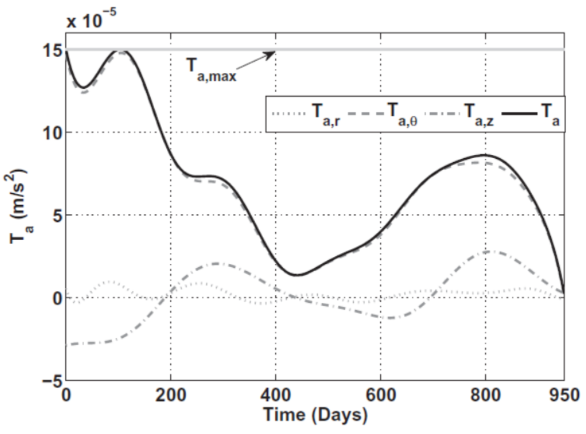


(b) The computed trajectory. The departure MJD2000 is 9604.5 and the time of flight is 850 days.

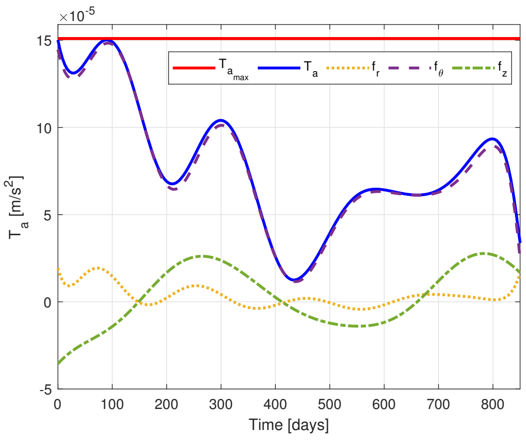
Figure 7.10: A comparison between the best trajectories computed by Taheri and Abdelkhalik [22] (left) and the author (right).

On the other hand, specifically looking at the radial (f_r) and axial (f_z) components separately, the resemblance of Figures 7.11a and 7.11b is even stronger. The axial acceleration shows the same trend and in both figures the radial thrust has a slight oscillation around the $0\,\mathrm{m/s^2}$ line, indicating the optimality of this solution, as f_r corresponds to gravity losses. Moreover, it makes sense that the transverse (f_θ) acceleration is significantly larger than the other two components at each epoch, as it is most efficient to apply thrust in the direction of flight.

The explanation of the slightly different thrust profiles can probably be found in the tolerances that are pushed through the solving algorithm. Figure 7.11b is found with a tolerance of 1×10^{-5} on the constraint, the fitness vector and the decision vector. Lowering this value will lead to a closer similarity between the three figure couples above, but this will be at the cost of computation time.



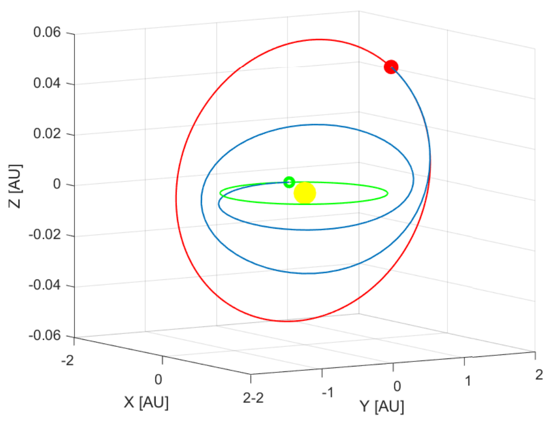
(a) The reference thrust profile corresponding to Figure 7.10a [22].



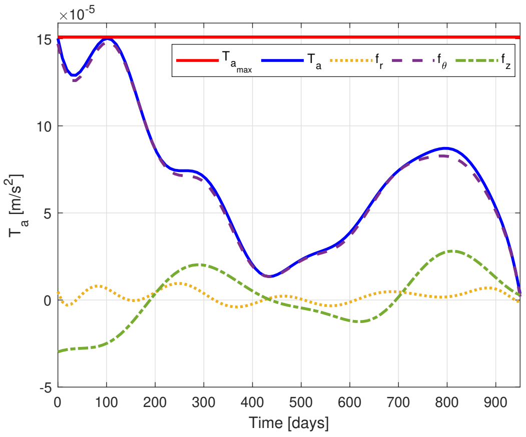
(b) The computed thrust profile corresponding to Figure 7.10b.

Figure 7.11: A comparison between the best thrust profiles computed by Taheri and Abdelkhalik [22] (left) and the author (right).

To prove the effect of lower tolerances, they all have been set at 1×10^{-9} and the simulation has been run again with the settings for the optimal trajectory as input data. The result of this run can be seen in Figure 7.12. Both the trajectory in Figure 7.12a and the thrust profile in Figure 7.12b show a high degree of similarity with the reference data in Figures 7.10a and 7.11a.



(a) The computed trajectory when the reference data as in Figure 7.10a is used directly



(b) The corresponding thrust profile to Figure 7.12a.

Figure 7.12: The separately computed transfer trajectory (left) and the corresponding thrust profile (right).

7.4. Tempel-1 in 3D

To gain a better insight in the compatibility of the finite Fourier series method with celestial objects which are in a more eccentric or more inclined orbit, the comet Tempel-1 was chosen as a target. As seen in Table 7.5, the eccentricity of the comet is 0.51159 and it is at an inclination of 10.5025° with respect to the ecliptic. In Figures 7.13 to 7.16 the outcome of the comparison of the results from Taheri and Abdelkhalik [22], Novak and Vasile [12] and the author will be given in the form of a porkchop plot, the three-dimensional trajectory and the corresponding thrust profile. The reference values to investigate this validation case and construct the unperturbed Tempel-1 orbit can be found in Tables 7.4 and 7.5.

Table 7.4: The input parameters that have been used to generate the trajectories from Earth to the comet Tempel-1 [22].

Input parameter	Value
$T_{a_{max}}$ [m/s^2]	7.1×10^{-4}
ΔT Departure date [days]	15
ΔT Time of flight [days]	20
Launch date range [MJD2000]	0.5 - 5845.5
Time of flight bounds [days]	400 - 1500
Points per revolution [-]	20
N_{rev} range [-]	0-2
n_r [-]	8
n_θ [-]	8
n_z [-]	6

Table 7.5: The Keplerian elements of the comet Tempel-1 that have been used to compute an unperturbed reference orbit [22].

Kepler element	Value
a [AU]	3.14009
e [-]	0.51159
i [deg]	10.5025
Ω [deg]	68.8818
ω [deg]	179.3031
M [deg]	203.23760
Epoch [MJD]	56717

The porkchop plots that have been generated for the comet Tempel-1 can be seen in Figure 7.13. It should be noted that Taheri and Abdelkhalik [22] did not present these results, but only the best trajectory and its matching thrust profile. The plot in Figure 7.13a has been generated with the spherical shaping method by Novak and Vasile [12]. Again, just as in Figure 7.8 the white areas in Figure 7.13a demonstrate infeasible trajectories, whose criteria were again not specified, but in Figure 7.13b all computed trajectory solutions have been included. The found ΔV values range from about 12 to 80 km/s. However, to better compare the results, the scale of Figure 7.13b has been adjusted such that it matches the one of Figure 7.13a.

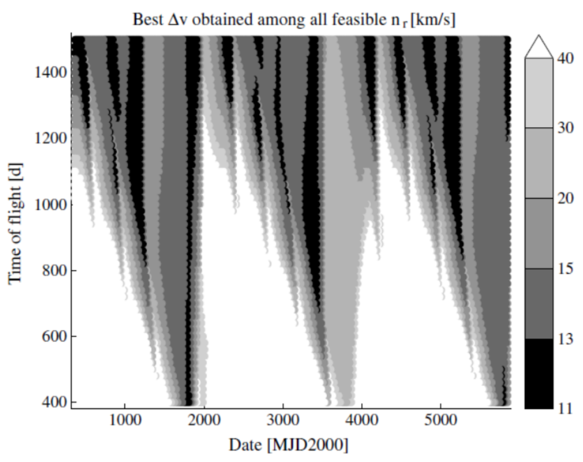
From Figure 7.13, the first conclusion that can be drawn is that by means of the finite Fourier series more feasible solutions can be found, thus offering more possibilities to be used as an input for direct solvers.

A second observation is the double period variation that is shown in the white triangular regions. The orbital period of the Earth is only 365 days, while Tempel-1 takes slightly over 2000 days to make one revolution around the Sun. In this time range from MJD2000 0.5 to MJD2000 5845.5 the comet can therefore make almost three revolutions, which correspond to the large white areas in the figure. The smaller orbital period of the Earth is responsible for the so-called subtriangles that can be observed. This means that the arrival position of Tempel-1 has more effect on the required ΔV than the departure position of the Earth. Due to the eccentricity of the orbit of Tempel-1, large velocity differences are induced over the entire trajectory. The periphelion is at a distance of 1.53 AU, while the aphelion is at a distance of 4.75 AU from the Sun.

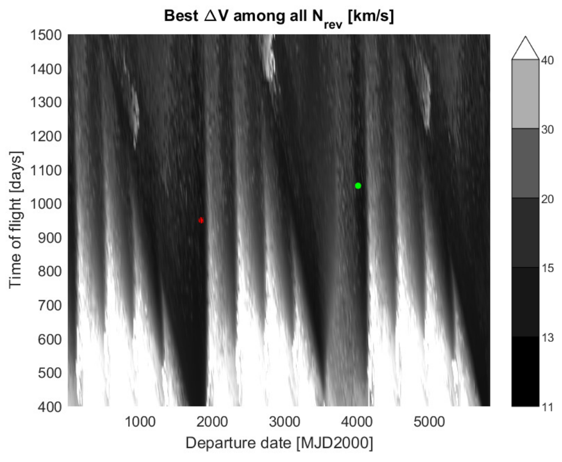
Finally, just as in the previously described case to Mars, the exact departure date and time of flight show a slight difference with respect to the least ΔV solution that was found by Taheri and Abdelkhalik [22]. The green dot in Figure 7.13b resembles the solution by Taheri and Abdelkhalik [22], while the red dot depicts the optimal trajectory the author has found. As the figure shows, the time of flight only differs by 100 days, which has a relatively small effect on the computed ΔV value. The departure date does however show a large difference of more than 2000 days. At first this might look like much, but as mentioned before, the orbital period of Tempel-1 is slightly over 2000 days. Knowing this, it can be said that the optimal solution is practically in the same region, except that it is shifted by one orbital period of the comet.

In Figure 7.14 two of the solutions from the infeasible regions have been shown. Apparently, both trajectories are mathematically feasible, but the necessary thrust acceleration cannot be accommodated by modern low-thrust engines, yet.

The difference in the departure date and the time of flight of the optimal trajectories by Taheri and Abdelkhalik [22] and the author were already mentioned before. In Figure 7.15 the two trajectories are shown and from here the discrepancy in the arrival position of the comet becomes clear. The large difference in the departure dates of both solutions is about 150 days, due to the periodic motion of the celestial bodies. Combined

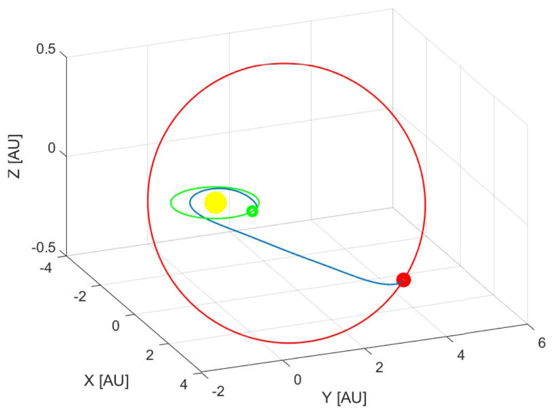


(a) Reference porkchop plot [12].

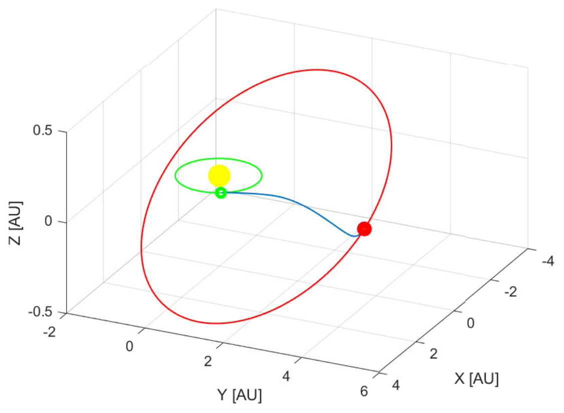


(b) Computed porkchop plot.

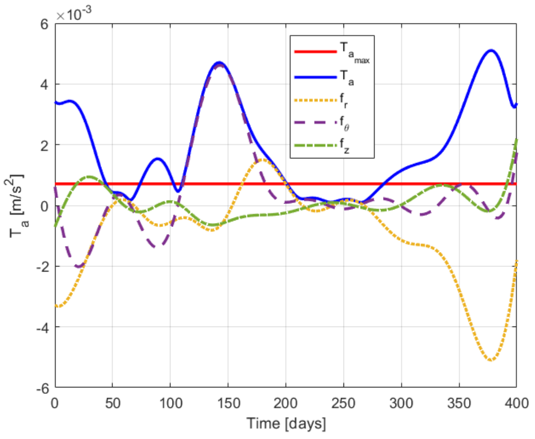
Figure 7.13: A comparison between the best porkchop plots computed by Novak and Vasile [12] (left) and the author (right).



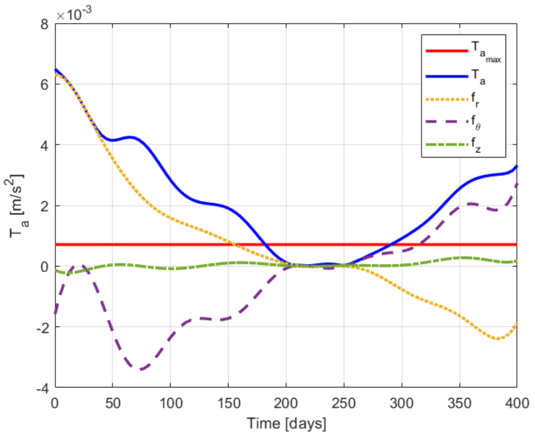
(a) The departure MJD2000 is 300
and the time of flight is 400 days.



(b) The departure MJD2000 is 4676.5
and the time of flight is 400 days.



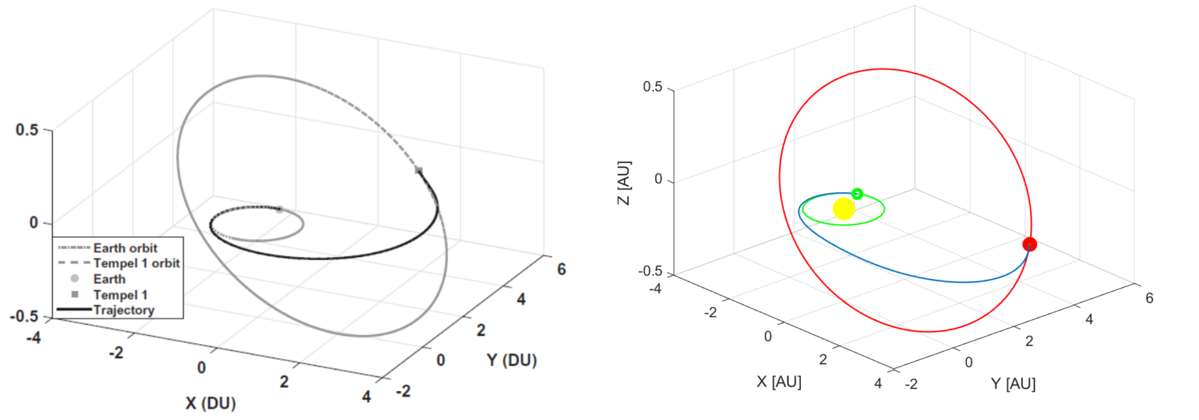
(c) The thrust profile corresponding to Figure 7.14a.



(d) The computed thrust profile corresponding to Figure 7.14b.

Figure 7.14: An example of two trajectory solutions and their thrust profiles that have been deemed infeasible.

with the reduced flight time of 100 days it does make sense that Tempel-1 is about 50 days back in its orbit, compared to the solution Taheri and Abdelkhalik [22] found.

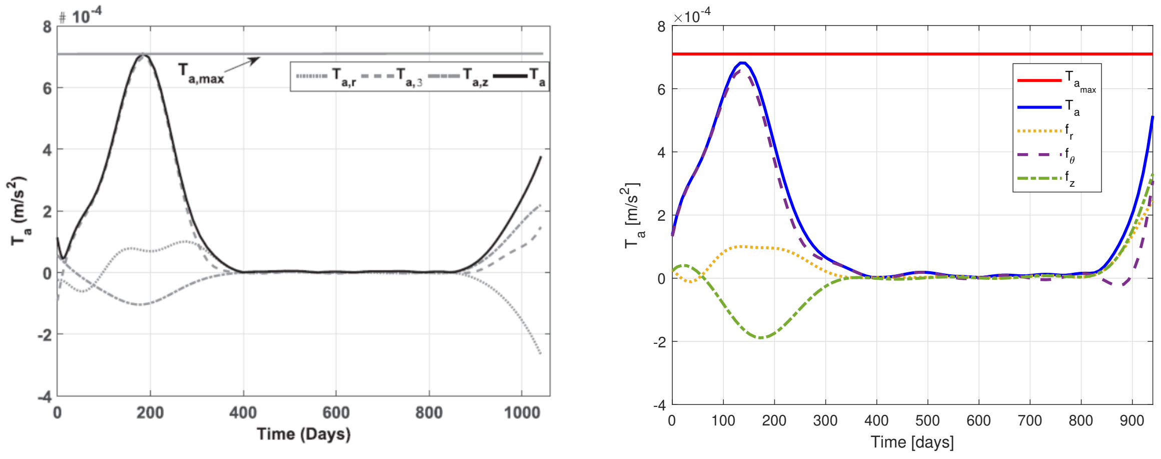


(a) The reference trajectory [22]. The departure MJD2000 is 4004.5 and the time of flight is 1040 days.

(b) The computed trajectory. The departure MJD2000 is 1845.5 and the time of flight is 940 days.

Figure 7.15: A comparison between the best trajectories computed by Taheri and Abdelkhalik [22] (left) and the author (right).

Figure 7.16 contains the thrust profiles that accompany the trajectories from Figure 7.15. Despite the fact that the trajectories might not fully agree with each other, the thrust profiles do show a larger resemblance. Both figures start with a rather steep increase in thrust after which it decreases and the spacecraft experiences a long period of about 450 days of (nearly) ballistic flight. At the end the engine power is throttled up again to fully match the orbit of Tempel-1. The only major difference can be seen in the behaviour of the radial acceleration component f_r. In Figure 7.16a it dives into the negative region, indicating a thrust force towards the Sun, because it needs to compensate for the small overshoot it has compared to the orbit of the comet. The radial acceleration is directed in the exact opposite direction in Figure 7.16b, as in this case the transfer orbit has no overshoot with respect to the target orbit and even needs a tiny bit of force to direct it towards the final destination. In the end, the difference in the behaviour of the acceleration can be explained by the small difference in arrival position between Figure 7.15a and Figure 7.15b.

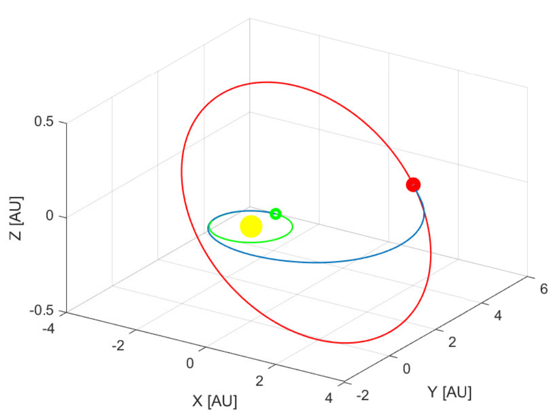


(a) The reference thrust profile corresponding to Figure 7.15a [22].

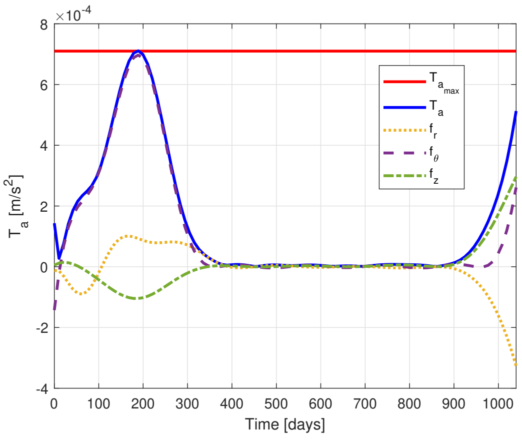
(b) The computed thrust profile corresponding to Figure 7.15b.

Figure 7.16: A comparison between the best thrust profiles computed by Taheri and Abdelkhalik [22] (left) and the author (right).

Just as the trajectory to Mars described in Section 7.3, the proof for the effects of lowered tolerances can be seen in Figure 7.17. This time, the tolerances have been set to 1×10^{-9} as well. On such a large distance scale, the similarity in arrival position between Figures 7.15a and 7.17a is somewhat harder to see, but the improved resemblance between the thrust profiles in Figures 7.16a and 7.17a is evident.



(a) The computed trajectory when the reference data as in Figure 7.15a is used directly



(b) The corresponding thrust profile to Figure 7.17a.

Figure 7.17: The separately computed transfer trajectory (left) and the corresponding thrust profile (right).

8 Model Development

Before the correct implementation method as described in Chapters 5 and 6 was figured out, some ambiguities had to be overcome when interpreting the discoveries by Taheri and Abdelkhalik [21, 22]. This chapter will explain these inconveniences and it will elaborate on possible additions to the method, starting with the definition of the decision vector in Section 8.1. The ambiguity of the unconstrained finite Fourier series is demonstrated in Section 8.2. Followed by that is the tuning of the solver in Section 8.3. The reference frame discrepancy is covered in Section 8.4 and Section 8.5 concludes the chapter with a sensitivity analysis concerning the number of Fourier terms that are used to model a transfer orbit.

8.1. Alternative Decision Vector

Before the right implementation method as described in Chapter 5 was found, another approach was taken. The first important difference is the definition of the decision vector. To recall Equation (5.17), the correct decision vector looks as follows:

$$\mathbf{x} = \begin{bmatrix} a_0 & a_3 & a_4 & \cdots & a_{n_r} & b_3 & b_4 & \cdots & b_{n_r} & c_0 & c_3 & c_4 & \cdots & c_{n_\theta} & d_3 & d_4 & \cdots & d_{n_\theta} \end{bmatrix} \tag{8.1}$$

It does not include the first two coefficients of each sine and cosine term in the Fourier series (i.e. a_1, a_2, b_1, b_2, c_1, c_2, d_1 and d_2), which means that its length is ($2n_r + 2n_\theta$ - 6). The erroneously implemented decision vector is displayed in Equation (8.2) below:

$$\mathbf{x} = \begin{bmatrix} a_0 & a_1 & a_2 & \cdots & a_{n_r} & b_1 & b_2 & \cdots & b_{n_r} & c_0 & c_1 & c_2 & \cdots & c_{n_\theta} & d_1 & d_2 & \cdots & d_{n_\theta} \end{bmatrix} \tag{8.2}$$

In this case, the total length of the decision vector is ($2n_r + 2n_\theta + 2$), as it includes the initial eight terms.

To ensure that the coefficients a_1, a_2, b_1, b_2, c_1, c_2, d_1 and d_2 are indeed equal to Equation (5.13a) through (5.13h), eight equality constraints had to be applied. The result of this run with the data from Table 7.2 can be seen in Figure 8.1.

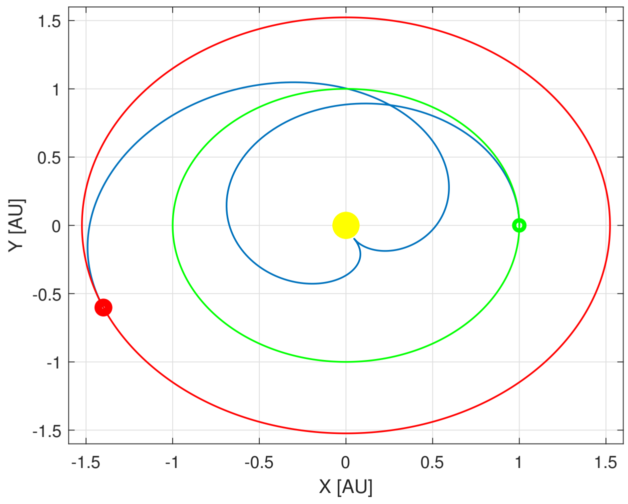


Figure 8.1: The found trajectory when only the first eight coefficients are enforced.

As can be observed, this caused an utterly unconventional trajectory shape. In addition to that, the corresponding ΔV is 157.5 km/s, which is multiple times the amount it would normally take in order to get to the red planet [12].

Because of these results, the solution space had to be restricted even more. For that reason it was decided that an inequality constraint would be applied to the orbital radius stating that the radius at the current DP should always be larger than at the previous DP. Depending on the number of discretisation points, this would entail that $(m-1)$ constraints are added to the problem, bringing the total number of constraints to $(m+7)$. This should prevent the trajectory from turning inward again. The result of this improvement is shown in Figure 8.2.

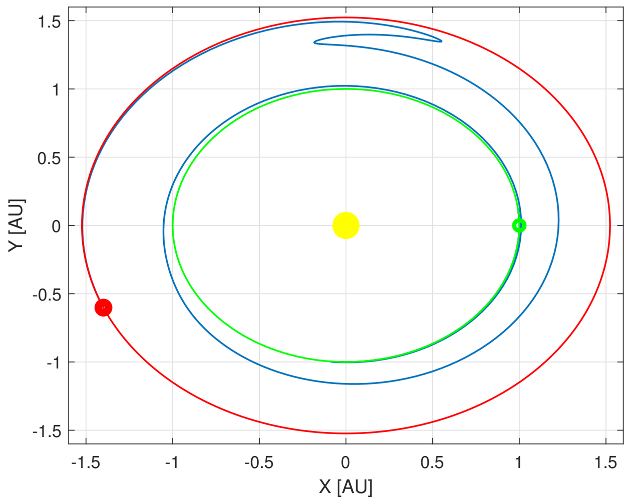
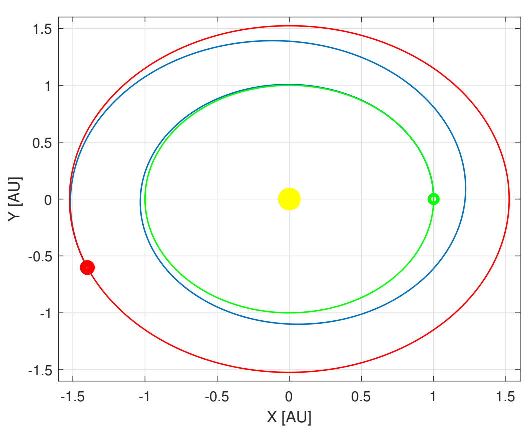


Figure 8.2: The found trajectory when the first eight coefficients and an increasing radius are enforced.

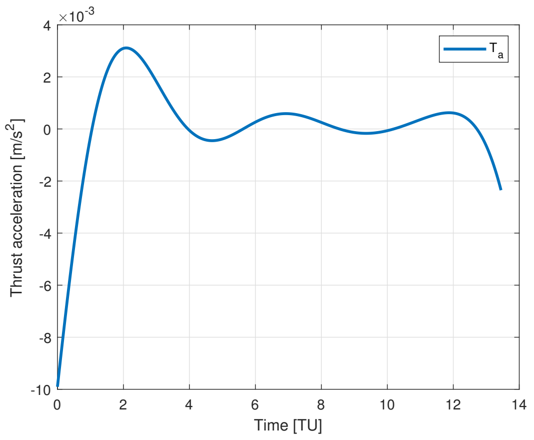
This trajectory clearly shows more similarity to Figure 7.3a, but it still contains some big flaws. The radius does increase over time, but when about 1.25 revolutions have been completed, two immediate inconsistencies show up, which explains the high ΔV of 543.1 km/s.

Again, this has proven to not yield the desired result. Therefore, another set of constraints was introduced. Due to the circular movement of an orbit, it makes sense that the transfer angle keeps on increasing when moving outward in the Solar System. Hence, the other set of implemented inequality constraints states that the current transfer angle should always be larger than the previous transfer angle. With this new set, another

$(m-1)$ constraints are added, bringing the total number of constraints to $(2m+6)$. The resulting trajectory is shown below in Figure 8.3a.



(a) The found trajectory when the first eight coefficients, an increasing radius and an increasing transfer angle are enforced.



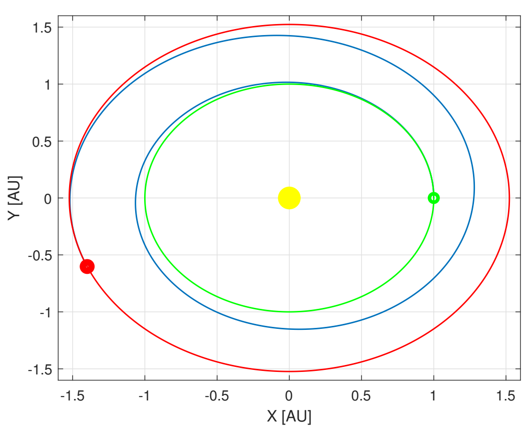
(b) The thrust profile including the constraints on the first eight coefficients, the radius and the transfer angle.

Figure 8.3: The trajectory solution and the corresponding thrust profile as a result of the imposed limits on the transfer angle.

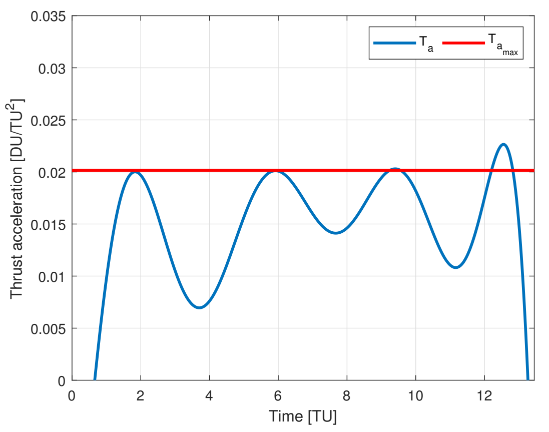
Comparing Figure 8.3a and Figure 7.3a, it is hard to see a difference, which is strange, as the trajecotory requires a ΔV of 67.8 km/s. For that reason, the thrust profile in Figure 8.3b needs to be analysed as well.

Looking at the figure, the explanation immediately becomes clear. For a better interpretation of the results, the acceleration has been expressed in m/s^2. Knowing that the constrained case, as in Table 7.2, uses a thrust constraint of 0.11 mm/s^2, this trajectory has a maximum acceleration of more than 3.11 mm/s^2, which is over 26x higher than the maximum acceleration the engine can handle.

In order to resolve these deviations, the thrust constraint has been turned on, resulting in the trajectory and thrust profile as seen in Figure 8.4.



(a) The found trajectory when the first eight coefficients, an increasing radius, the transfer angle and the thrust are enforced.



(b) The thrust profile including the constraints on the first eight coefficients, the radius, the transfer angle and the thrust.

Figure 8.4: The trajectory solution and the corresponding thrust profile as a result of the activation of the thrust constraint.

The transfer orbit in Figure 8.4a is identical to the one in Figure 8.3a, and thus to the original one by Taheri and Abdelkhalik [21] in Figure 7.3a. However, this does not hold for the thrust profile. The thrust constraint manages to keep the acceleration much closer to the limit of 0.02 DU/TU^2, but it cannot totally prevent it from exceeding this limit. The required ΔV for this mission is only 6.08 km/s, which makes much more sense than the values found before. Nonetheless, the thrust profile does not match Figure 7.3b at all, hence it was concluded that this is not the proper way the finite Fourier series method had to be implemented.

8.2. 2D Unconstrained Finite Fourier Series

When the first test run with the unconstrained finite Fourier series method was done, some peculiar behaviour was observed. The trajectory to Mars has been set up with the exact same settings as in Section 8.1, but this time the thrust acceleration was not limited. The trajectory solution that is obtained is plotted in Figure 8.5.

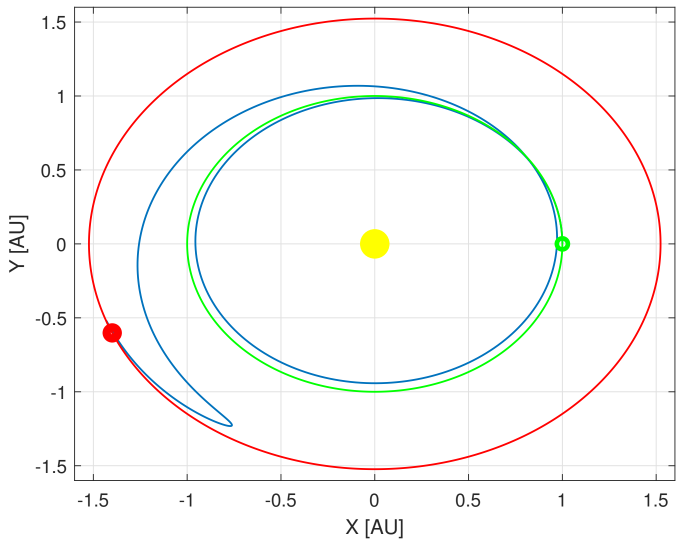


Figure 8.5: The result of the first run obtained with the unconstrained finite Fourier series.

The required ΔV for this trajectory is a whopping 125.5 km/s, which is far beyond the nominal value of about 5.6 to 5.8 km/s. The reason why it is so high can be attributed to the sudden change in direction near the end of the trajectory. This directional change requires the spacecraft to kill all its velocity and move into the opposite direction, rendering the solution infeasible.

The problem was solved by adding additional constraints on the transfer angle. It is prescribed that the transfer angle at DP n should always be larger than the transfer angle at DP $(n-1)$, ensuring an orbit shape that always continues in the same direction, similar to the method mentioned in Section 8.2. This also means that the name *unconstrained finite Fourier series* only refers to the absence of the thrust constraint. The total number of constraints on the transfer angle depends on the number of discretisation points and can be described as $(\#DP-1)$.

8.3. Solver Tuning

Using a numerical solver is not as easy as selecting it and feeding it with the problem. It has to be tuned to the specific problem it is solving. The problem concerning deviating ΔV values is explained in Section 8.3.1, after which the solution is explained in Section 8.3.2.

8.3.1. ΔV Deviations

As soon as the problem with the redefinition of the decision vector was solved, another interesting result was found. The validation case as described in Section 7.2 exactly behaved as it should, hence conforming the effectiveness of the finite Fourier series method.

However, when the same problem was fed into the solver with a different number of discretisation points, a new issue showed up. For some DPs the ΔV shot up to excessively high values. This seems to happen at random. Figure 8.6 shows the results of this analysis on a range from 15 to 80 DPs.

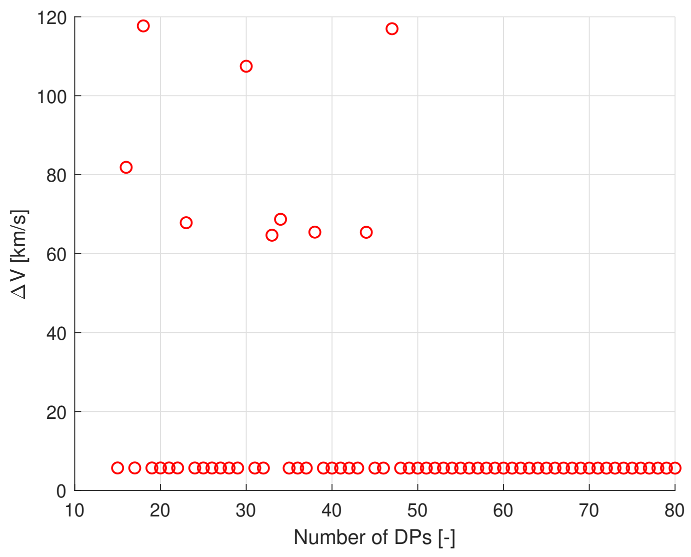
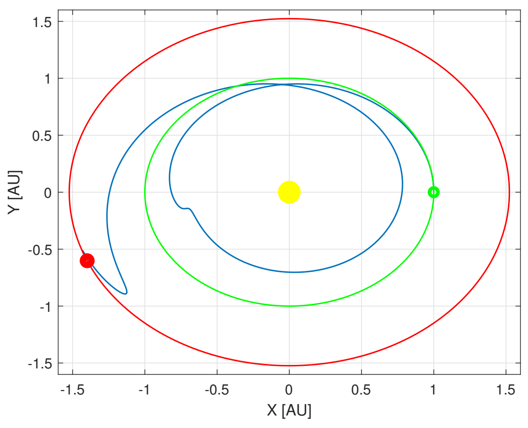


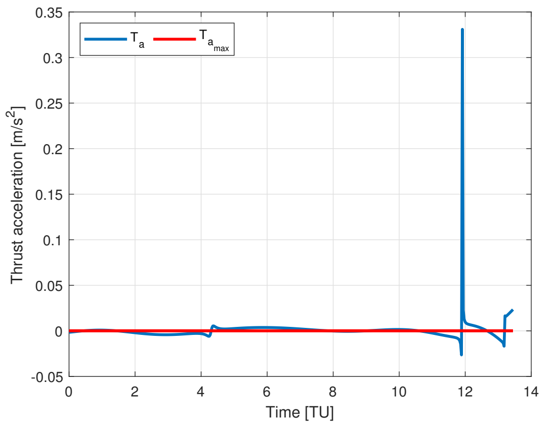
Figure 8.6: Excessively high ΔV values are found for some random DPs.

If the corresponding trajectories and thrust profiles are plotted, the results can be explained. This has been done for the case with 33 DPs and a ΔV of 64.66 km/s in Figures 8.7a and 8.7b respectively. It can be seen that the trajectory has a peculiar zigzag-shape at the very end, which could be seen in Figure 8.5 as well. This zigzag indicates that suddenly all velocity needs to be killed, as it moves in the complete opposite direction for a short period of time, after which it finds its way back to the correct destination orbit.

In the thrust profile in Figure 8.7b this effect is clearly standing out too. At the end, a large upward peak is found that indicates much thrust is given in the direction of the current movement, probably to match the orbital velocity of Mars.



(a) An example of an erroneous trajectory.



(b) The corresponding thrust profile.

Figure 8.7: The incorrect trajectory solution and the corresponding thrust profile when 33 DPs are used.

8.3.2. Random Variables

In order to solve the problem that was encountered and described in Section 8.3.1, a different approach had to be taken. The original idea was to take the initial guess found from the cubic polynomial in Equation (4.19) and feed it into the solver as a starting point as prescribed by Taheri and Abdelkhalik [21]. Nonetheless this led to the results in Figure 8.6, which are probably caused by the Nelder-Mead algorithm getting stuck in a local minimum. Therefore, it was decided to include a random element in the solving procedure to push it into a different direction to continue its search.

It was decided that the numbers the initial guess vector comprised of would be used as the baseline for a normal distribution. This means that with these values a mean and a variance have been calculated. Then, if the solver is not able to find a possible solution with the given initial guess, a random number from this normal distribution is added to each separate number in the vector such that the search can be resumed.

The initial guess vector for the problem described in Section 7.2 is presented in Table 8.1. With this data, a mean μ of 1.44 and a variance σ^2 of 6.55 are found, from which a normal distribution can be constructed. The multi-start initial guesses will then be based on this distribution.

Table 8.1: The initial guess based on the case described in Table 7.2.

Coefficient	a_0	c_0	c_3	c_4	c_5	d_3	d_4	d_5
Value	2.529	0	4.68	0.882	-0.77	3.82	3.17	-2.83

The results of a new run in which the number of DPs is increased up to 500 are shown in Figure 8.8. In here, the expected asymptotic behaviour Taheri and Abdelkhalik [21] talked about can be clearly observed.

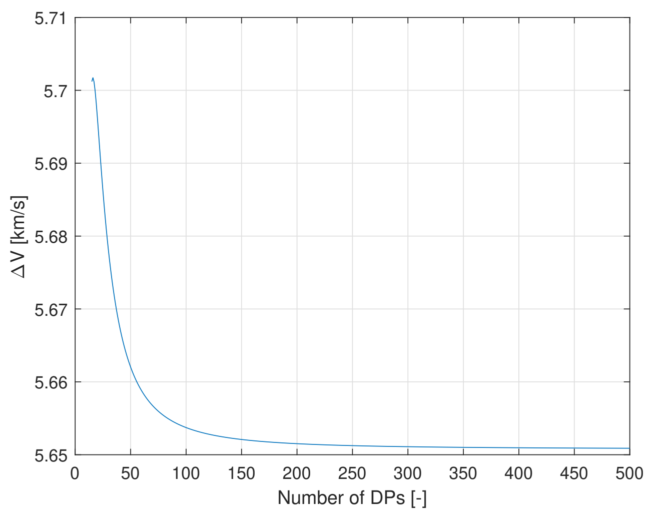


Figure 8.8: After the multi-start feature has been implemented, the ΔV behaves as it should for an increasing number of discretisation points.

8.4. Reference Frame

The second step in setting up a finite Fourier series trajectory model is to define the boundary conditions. The following equation is given in order to compute the transfer angle θ_f at the position of the arrival body (explaining the subscript f) as a function of the number of revolutions N_{rev} [21]:

$$\theta_f = \theta_0 + N_{\text{rev}} \cdot 2\pi \tag{8.3}$$

in which θ_0 represents the initial angle between the departure body at the start ($t = 0$) and the arrival body at the end ($t = T$), measured counter-clockwise, which is illustrated by Figure 8.9.

Taheri and Abdelkhalik [21] however did not mention that the transfer angle is always normalised in such a way that the positive x-axis consistently serves as the datum line from which θ is measured. This causes problems when the SPICE libraries are used to obtain the state vectors of celestial bodies with respect to the ecliptic reference frame, as this normalisation is not accounted for.

In order to solve it, two things can be done: either the state vector is corrected for this normalisation, or Equation (8.3) is slightly altered. The approach that has been followed in this implementation as explained in Chapters 5 and 6 is the latter one, for which the initial transfer angle θ_i is added to Equation (8.3), resulting in Equation (8.4).

$$\theta_f = \theta_0 + N_{\text{rev}} \cdot 2\pi + \theta_i \tag{8.4}$$

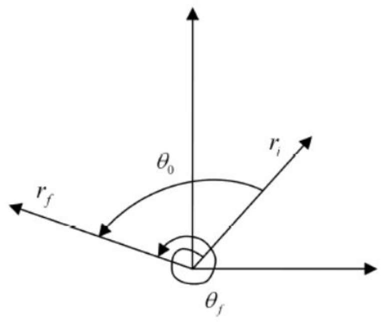


Figure 8.9: A graphical representation explaining how the initial angle θ_0 between the departure body at $t = 0$ and the arrival body at $t = T$ is found.

8.5. The Number of Fourier Coefficients

The finite Fourier series method allows its user to set the number of Fourier terms that are used to capture a trajectory. This can be done in all dimensions and usually a certain degree of complexity asks for certain settings. Taheri and Abdelkhalik [22], for example, used the following number of terms for a trajectory to Mars (low eccentricity and inclination): $n_r = 6$, $n_\theta = 6$ and $n_z = 4$, while a more challenging target, like the comet Tempel-1 (high eccentricity and inclination) requires: $n_r = 8$, $n_\theta = 8$ and $n_z = 6$. A different number of Fourier series will influence the shape and thus the required ΔV for a certain candidate trajectory.

To show this, a trajectory to Jupiter has been computed. The parameters that have been used to generate the transfer orbit are shown in Table 8.2, while the oribital elements that have been used to set up the orbit of Jupiter are stated in Table 8.3. The corresponding trajectory solution is depicted in Figure 8.10.

Table 8.2: The input parameters that have been used to generate the reference trajectory from Earth to Jupiter [13].

Input parameter	Value
Departure date [MJD2000]	6653
Time of flight lower [years]	4
Points per revolution [-]	100
N_{rev} [-]	1
n_r [-]	6
n_θ [-]	6
n_z [-]	4
Tolerances [-]	1×10^{-9}
ΔV [km/s]	17.30

Table 8.3: The orbital elements of Jupiter that have been used to compute an unperturbed reference orbit [10].

Kepler element	Value
a [AU]	5.2
e [-]	0.048
i [deg]	1.304
Ω [deg]	100.5
ω [deg]	274.25
M [deg]	19.66796
Epoch [MJD]	51544.5

To examine the influence of the number of Fourier coefficients on the trajectory solutions, the following approach was taken: each time a run was done, one Fourier term was added to one of the dimensions while the others were kept at their standard value as in Table 8.2. This means that in total nearly 300 different trajectories were computed. Figure 8.11 shows the range of possible trajectories that are generated due to this large variety in Fourier coefficients.

As can be seen in the figure, there is an especially clear difference in the axial position at a certain point in time of the various trajectories. To illustrate what happens when the number of Fourier terms is set too high for a certain trajectory design problem, one case was highlighted. The outcome of this case, where n_r and n_θ are kept at their values as written in Table 8.2 whilst n_z was set at 25, is displayed in Figure 8.13.

The random periodic behaviour can be observed in every aspect of the transfer orbit. The overall trajectory shape in Figure 8.13a already shows some surprising bumps. When the orbit is inspected in the zy-plane, as in Figure 8.13b, this effect is even more evident. The isolated axial component in Figure 8.13c shows the sinusoidal waves especially in the first half of the flight, but the effects are best witnessed in Figure 8.13d, where the three components of the thrust are depicted together with the total thrust. The $z-$component shows an

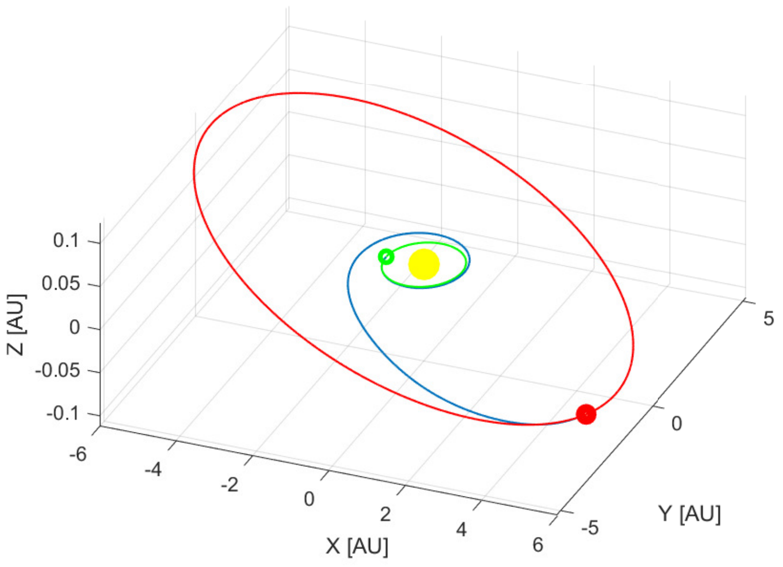


Figure 8.10: The reference trajectory to Jupiter [10].

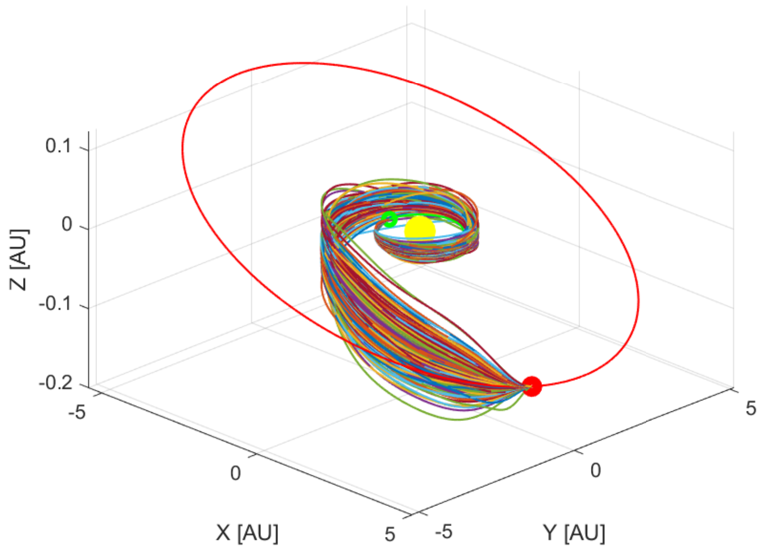


Figure 8.11: An overview of the trajectories that are found when the number of Fourier coefficiens is increased up to 100 per dimension.

entirely sinusoidal wave pattern, which has its effects on the total thrust acceleration as well, specifically in the first 400 days of the trajectory.

Another intriguing result was found when the final outcome in terms of ΔV is plotted against the number of used coefficients. The impact caused by using more coefficients for the Fourier series describing r and θ seems to be following the same pattern. However, this cannot be said for the axial Fourier series. Based on Figure 8.12 it can be concluded that the obtained trajectory solution is most sensitive to the number of Fourier coefficients that will be used to describe the movement in z.

Furthermore, the general trend that can be observed in Figure 8.12 prescribes that the ΔV increases the more coefficients are used beyond a certain threshold, as the minimum ΔV is achieved with 18 terms for n_r. However, knowing that an increase of n terms will extend the complexity of the problem with an equal

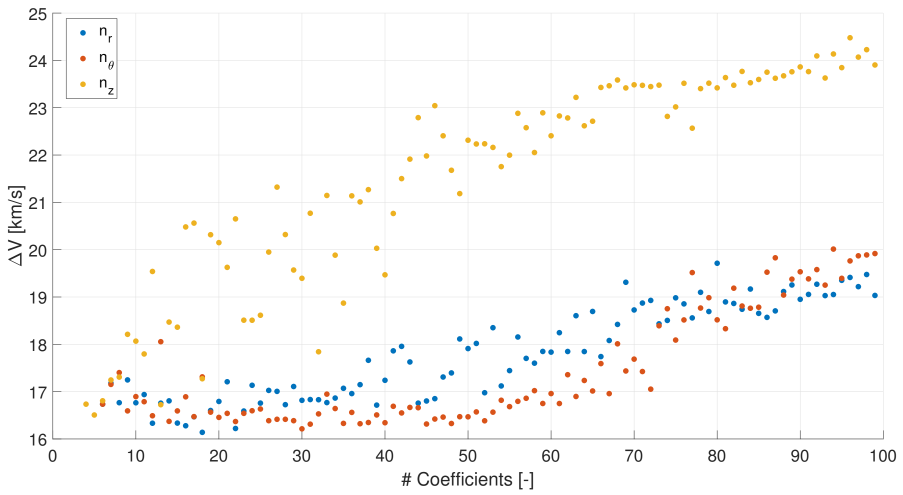
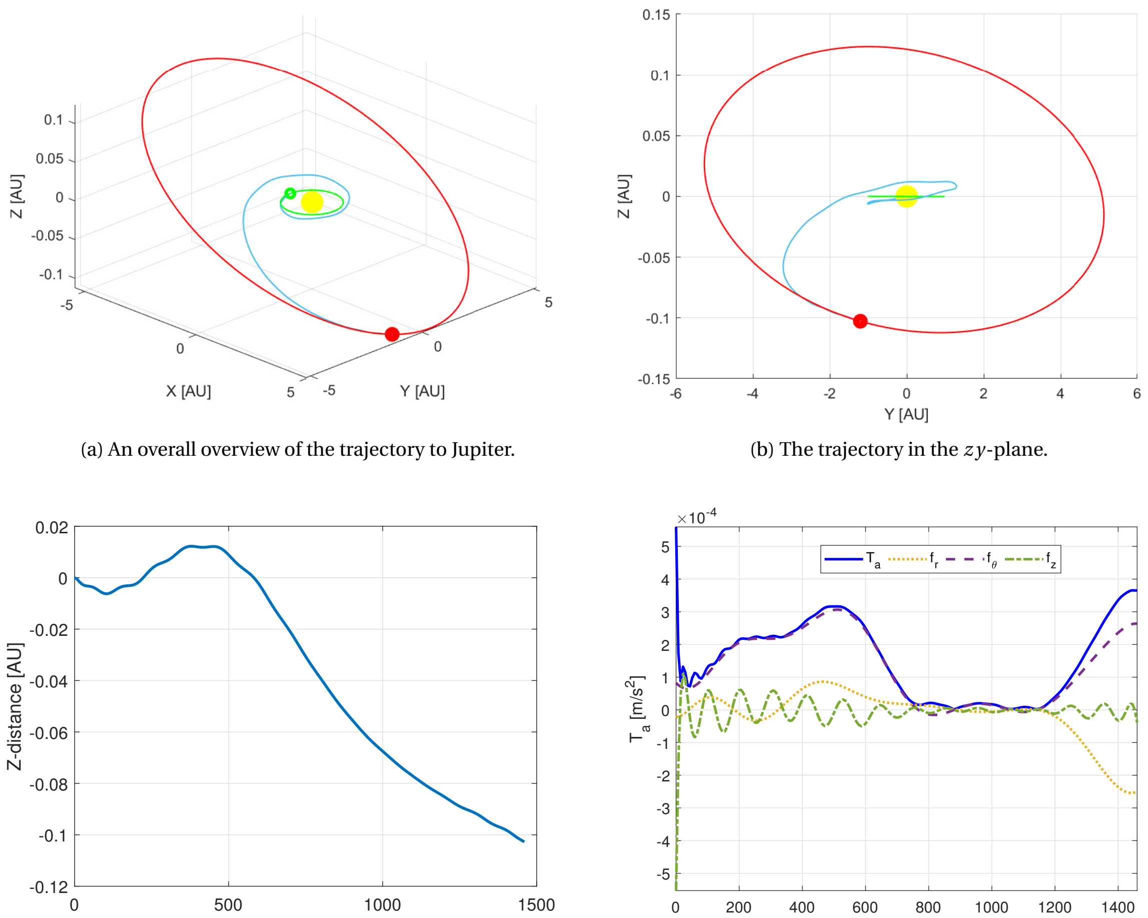


Figure 8.12: The increase in n_z leads to a significantly higher increment regarding the ΔV compared to the increase of n_r and n_θ.

number of dimensions and thus it will require more computation time. Therefore, it should always be striven for to have the least number of Fourier terms.



(a) An overall overview of the trajectory to Jupiter.

(b) The trajectory in the zy-plane.

(c) The behaviour of the axial coordiante.

(d) The thrust profile.

Figure 8.13: The effects of a high number of Fourier coefficients in the axial direction ($n_z = 25$).

III

Initialisation

9 Strategy

Having proven that the method functions properly, it can finally be used to do some additional research on its performance and its initialisation strategy in particular. In this chapter, the concept of initialisation is shortly summarised in Section 9.1, after which the steps that have been taking during the research process are elaborated upon. The test functions are introduced in Section 9.2, the test objects can be found in Section 9.3 and finally the five analysis cases are explicated in Section 9.4.

9.1. Finding the Optimum

In Section 4.3 it is explained that the Fourier coefficients for a certain trajectory are obtained by solving a non-linear programming problem. The fact that it is non-linear is crucial information here, as this entails that the solving algorithm requires some a priori information in order to find the solution. It is at this point where the so-called initialisation strategy comes into play. That strategy determines how these a priori values are determined and thus it strongly influences the direction the solver moves into.

This effect is nicely seen in Figure 9.1. The graph shows the various ΔV regions for a transfer trajectory from Earth to Uranus that incorporates a deep space manoeuvre, while taking advantage of the gravity-assists from Mars and Jupiter. As usual in the field of trajectory design, the solution that requires the least amount of ΔV is sought after, hence in this case it is preferred that the initialisation strategy directs the solver towards the deep blue area.

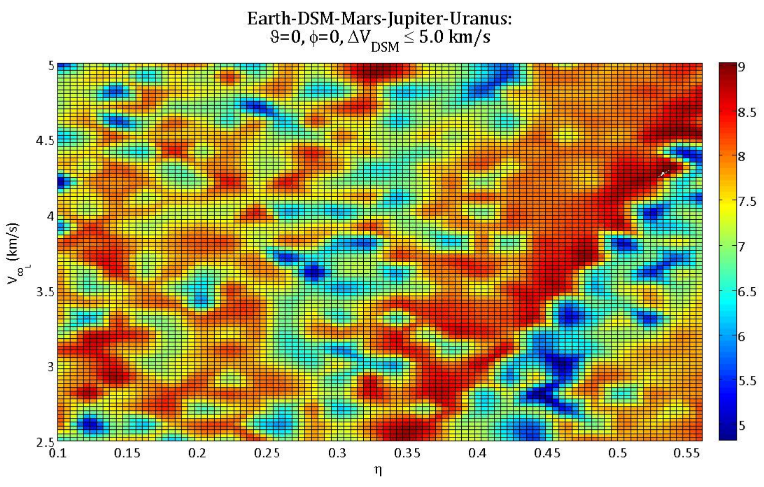


Figure 9.1: An example of a solution region with its various areas with minima and maxima [11].

In the original design of the finite Fourier series, a cubic polynomial function is used as an approximation for the trajectory from which the initial guess is derived. Furthermore, Taheri and Abdelkhalik [22] state that the initial unknown Fourier coefficients of the axial coordinate, z, are set to zero in all of the cases. The goal of this research is to investigate whether a better initialisation method exists and if its three-dimensional stability can be improved.

9.2. Function Types

This section contains an explanation of the four different initialisation functions and the derivations of their coefficients. First, the reference case consisting of a power function is presented in Section 9.2.1. After that, the newly designed exponential, trigonometric and logarithmic functions follow in Sections 9.2.2 to 9.2.4, respectively. To conclude, Section 9.2.5 correspondingly comments on the use of an exponential function combined with a power sine function. In Figure 9.2 a graphical overview of the all functions with various coefficients can be found.

The initialisation function is denoted by $f(\tau)$ and the coefficients of this function are determined by the boundary conditions of the trajectory design problem. Note that in this chapter only the radial distance r will be used to show the derivations of the function coefficients, but naturally the same rules apply to the transfer angle θ and the axial position z too. The boundary conditions for r are established as in Section 4.3, but they are stated here again for clarity reasons in Equations (9.1) and (9.2):

$$\begin{aligned} f(t=0) &= r_i \qquad & f(t=T) &= r_f \\ \dot{f}(t=0) &= \dot{r}_i \qquad & \dot{f}(t=T) &= \dot{r}_f \end{aligned} \tag{9.1}$$

$$\begin{aligned} f(\tau=0) &= r_i \qquad & f(\tau=1) &= r_f \\ f'(\tau=0) &= r'_i \qquad & f'(\tau=1) &= r'_f \end{aligned} \tag{9.2}$$

In the upcoming derivations of the initialisation function coefficient, the more complex three-dimensional derivation will be presented first, after which the value of the two-dimensional coefficients is shown.

9.2.1. Power Function

The initialisation functions that are currently used by Taheri and Abdelkhalik [22] are power functions of the third order. Their derivation and application have already been explained in Section 4.3 and chapters 5 and 6. The derivation of the coefficients can be found in Appendix A. Yet, for the sake of completeness, the functions are stated below:

$$f(t) = at^3 + bt^2 + ct + d \tag{9.3}$$

$$f(\tau) = a\tau^3 + b\tau^2 + c\tau + d \tag{9.4}$$

9.2.2. Exponential Function

Another proposed function that the solver can use as a reference value is an exponential function. As seen in Figure 9.2, its behaviour is similar to that of the previously mentioned function, but it could provide a more valuable estimation of the Fourier coefficients when the trajectory solution contains a rapidly increasing position parameter, for example. When approximating the trajectory as an exponential function, the used initialisation functions have the following form and contain only three coefficients:

$$f(t) = ae^{bt} + c \tag{9.5}$$

$$f(\tau) = ae^{b\tau} + c \tag{9.6}$$

To find those three coefficients the derivative of Equation (9.6) is needed:

$$f(\tau) = abe^{b\tau} \tag{9.7}$$

If the boundary conditions as in Equation (9.2) are substituted in Equation (9.7), the following two relations are obtained:

$$\begin{cases} f'(0) = ab = r'_i \\ f'(1) = abe^b = r'_f \end{cases} \tag{9.8}$$

The two constants a and b are removed by dividing the two expressions in Equation (9.8) such that only the exponent is left, resulting in:

$$\frac{f'(1)}{f'(0)} = \frac{abe^b}{ab} = e^b = \frac{r'_f}{r'_i} \tag{9.9}$$

From Equation (9.9) the value for b is easily computed according to:

$$b = \ln\left(\frac{r'_f}{r'_i}\right) \tag{9.10}$$

To solve for the other two coefficients a and c another system of equations is set up:

$$\begin{cases} f(0) = a + c = r_i \\ f(1) = ae^b + c = r_f \end{cases} \tag{9.11}$$

Substracting the equations from each other eliminates c such that the system can be solved for a:

$$a = \frac{r_i - r_f}{1 - e^b} \tag{9.12}$$

which can be further simplified by substituting the found solution for b from Equation (9.9):

$$a = \frac{r'_i\left(r_i - r_f\right)}{r'_i - r'_f} \tag{9.13}$$

By substituting Equation (9.13) in Equation (9.11) the final parameter c can be determined:

$$c = \frac{r'_i r_f - r_i r'_f}{r'_i - r'_f} \tag{9.14}$$

Note that in this derivation only the three-dimensional function was used. The two-dimensional coefficients are found in the exact same way, but it is just the regular time t that is used instead of the scaled time τ. This does not affect coefficients a and c, as to convert them to two-dimensional coefficients, only the primes need to be replaced by dots. For the b coefficient the scaling by T does however need to be accounted for. In Equations (9.15) to (9.17) the two-dimensional expressions for the coefficients are listed:

$$a = \frac{\dot{r}_i\left(r_i - r_f\right)}{\dot{r}_i - \dot{r}_f} \tag{9.15}$$

$$b = \frac{1}{T}\ln\left(\frac{\dot{r}_f}{\dot{r}_i}\right) \tag{9.16}$$

$$c = \frac{\dot{r}_i r_f - r_i \dot{r}_f}{\dot{r}_i - \dot{r}_f} \tag{9.17}$$

9.2.3. Trigonometric Function

A particular characteristic of low-thrust propulsion trajectories is their periodic circular motion (see Figure 9.2). The type of function that perfectly models this kind of behaviour would be a sinusoidal function. Another reason to choose this function type is the close relation to the underlying theory of Fourier series, which are sequences of sinusoidal functions. After all, the sine functions are modelled as follows:

$$f(t) = A\sin\left(\omega\left(t - \phi\right)\right) + K \tag{9.18}$$

$$f(\tau) = A\sin\left(\omega\left(\tau - \phi\right)\right) + K \tag{9.19}$$

in which the different parameters have a more tangible meaning compared to the two previously discussed initialisation functions. The amplitude of the sine function is described by A, while the frequency and the phase shift are denoted by ω and ϕ respectively. Finally, the K parameter indicates the mean of the curve.

The mean K is immediately found, as it is simply the mean of r_i and r_f. This holds as well for the amplitude A, because it is the maximum value of the difference between $f(\tau)$ and K. Subsequently, the other two parameters are found by rewriting Equation (9.19) into:

$$\omega\left(\tau-\phi\right)=\arcsin\left(\frac{f(\tau)-K}{A}\right) \tag{9.20}$$

which can be rewritten for τ:

$$\tau=\frac{1}{\omega}\zeta+\phi \tag{9.21}$$

in which ζ equals $\arcsin\left(\frac{f(\tau)-K}{A}\right)$. By evaluating Equation (9.21) at the two boundary conditions from Equation (9.2), a linear system of the following form can be set up:

$$\begin{bmatrix}\zeta(\tau=0) & 1\\ \zeta(\tau=1) & 1\end{bmatrix}\begin{bmatrix}\frac{1}{\omega}\\ \phi\end{bmatrix}=\begin{bmatrix}0\\ 1\end{bmatrix} \tag{9.22}$$

from where ϕ is found directly, while ω is obtained by inverting the outcome. Note however that the equation slightly changes in case a sine function is used to approximate a two-dimensional Fourier series, but the procedure to obtain K and A is identical. If Equation (9.22) is evaluated in the regular time domain, it has the following shape:

$$\begin{bmatrix}\zeta(t=0) & 1\\ \zeta(t=T) & 1\end{bmatrix}\begin{bmatrix}\frac{1}{\omega}\\ \phi\end{bmatrix}=\begin{bmatrix}0\\ T\end{bmatrix} \tag{9.23}$$

9.2.4. Logarithmic Function

The last initialisation function that will be tested is a logarithmic one. This type of function, which could be regarded as the inverse of the exponential function discussed in Section 9.2.2, could provide improved a priori values for trajectories that show a rapid change in the beginning after which a long zero-thrust arc follows, which can be seen in Figure 9.2. The logarithmic approximation that is used has the following expression:

$$f(t)=a+b\ln(t) \tag{9.24}$$

$$f(\tau)=a+b\ln(\tau) \tag{9.25}$$

With only two coefficients the system of equations that is obtained by substituting Equation (9.2) leads to:

$$f(1)=a=r_f \tag{9.26}$$

$$f'(1)=b=r'_f \tag{9.27}$$

which results in a taking the value of r_f and b taking the value of r'_f. In case of a two-dimensional approximation, the b coefficient is equal to $\dot{r}_f$ instead of r'_f.

9.2.5. Other Possible Function Types

For the design of more complex trajectories, a single periodic function that shows the same pattern over the entire time of flight, as with a trigonometric function, might not be enough. Therefore an exponential sinusoid (or damped sine wave) could provide a possible better estimation of the parameters. The exponential part allows for a change in amplitude and therefore a change in the overall function shape. Besides exponential terms, the amplitude can also be modelled by a power term, for example. In Figure 9.2 this function shape is depicted. The exponential or power term is combined with a sine function as follows:

$$e^{n\tau}\sin(2\pi m\tau) \tag{9.28}$$

$$\tau^n \sin(2\pi m\tau) \tag{9.29}$$

However, it has been found that the estimation of the different initialisation function parameters that construct an approximation of the trajectory takes up too much time. This is caused by the underlying data fitting method, which requires an initial estimation of the parameters as well [1]. Due to the added computational time and the reasonably added complexity of this problem, it has been decided to omit this function type from the current analysis.

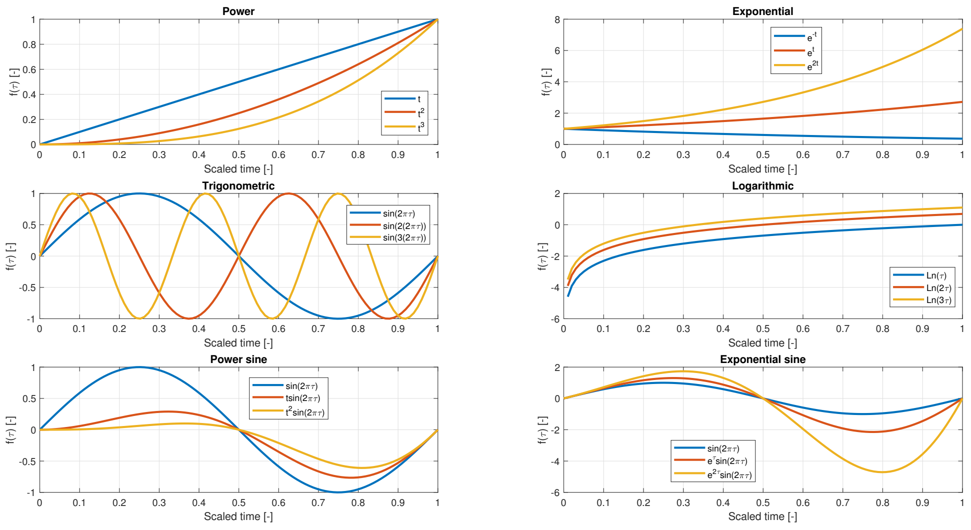


Figure 9.2: The behaviour of the six different initialisation functions.

9.3. Test Objects

The performance of the new initialisation functions will be assessed by means of transfer trajectories to a number of celestial bodies. Three different bodies have been picked to analyse their suitability: the planet Jupiter, the asteroid Dionysus and the dwarf planet Haumea, which are laid out in Sections 9.3.1 to 9.3.3, respectively. For Jupiter and Dionysus both a two-dimensional and a three-dimensional analysis have been included in order to assess the performance of the new initialisation strategies in both dimensions. This has not been done for Haumea, as a two-dimensional analysis would be of no use, considering its high inclination.

9.3.1. Jupiter

The first target that will be used to test the different functions will be Jupiter. Its orbit is determined by relatively safe parameters, as it has an eccentricity of 0.048 and an inclination of only 1.304° with respect to the ecliptic, which renders it a perfect example for the two-dimensional approach as well. This means that the only challenging element is the semi-major axis. The nearly circular and two-dimensional orbit make this planet a perfect target to analyse the behaviour of the simplest trajectory design problem. All Keplerian elements of Jupiter can be found in Table 9.1. The input parameters and reference values of the transfer trajectories are listed in Tables 9.2 and 9.3.

Table 9.1: The Keplerian elements of Jupiter that have been used to compute an unperturbed reference orbit [10].

Kepler element	Value
a [AU]	5.2
e [-]	0.048
i [deg]	1.304
Ω [deg]	100.5
ω [deg]	274.25
M [deg]	19.66796
Epoch [MJD]	51544.5

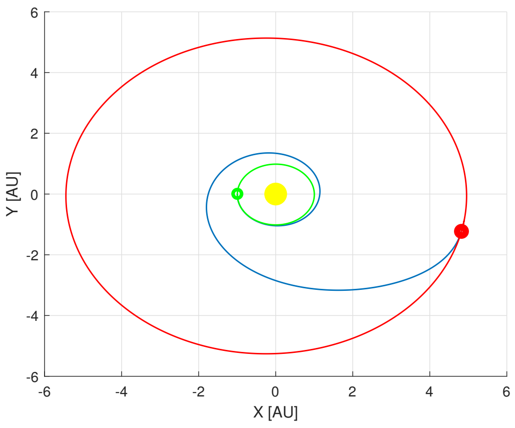
Table 9.2: The input parameters that have been used to generate the 2D reference trajectory from Earth to Jupiter [13].

Input parameter	Value
Departure date [MJD2000]	6653
Time of flight [years]	4
Discretisation points [-]	25
N_{rev} [-]	1
n_r [-]	2
n_θ [-]	5
Tolerances [-]	1×10^{-9}
Multi-start seed [-]	123
ΔV [km/s]	18.22
$T_{a_{max}}$ [mm/s^2]	0.25

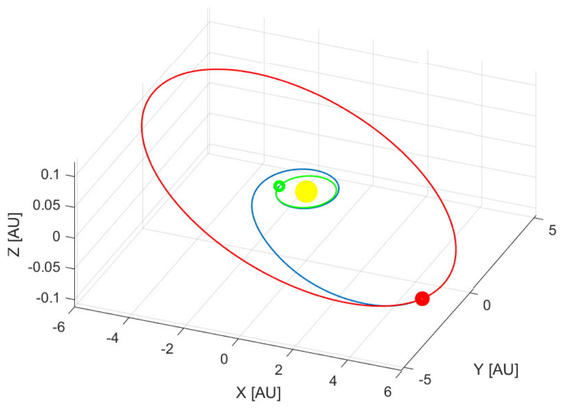
Table 9.3: The input parameters that have been used to generate the 3D reference trajectory from Earth to Jupiter [13].

Input parameter	Value
Departure date [MJD2000]	6653
Time of flight [years]	4
Points per revolution [-]	100
N_{rev} [-]	1
n_r [-]	6
n_θ [-]	6
n_z [-]	4
Tolerances [-]	1×10^{-9}
Multi-start seed [-]	123
ΔV [km/s]	17.30

The reference trajectories are plotted in Figure 9.3 and show a smooth spiralling path of motion. As can be seen in Figure 9.3a, the two-dimensional orbit is nearly identical to the three-dimensional one in Figure 9.3b, as the axial movement from the latter one barely reaches 0.1 AU.



(a) The 2D reference trajectory to Jupiter.



(b) The 3D reference trajectory to Jupiter.

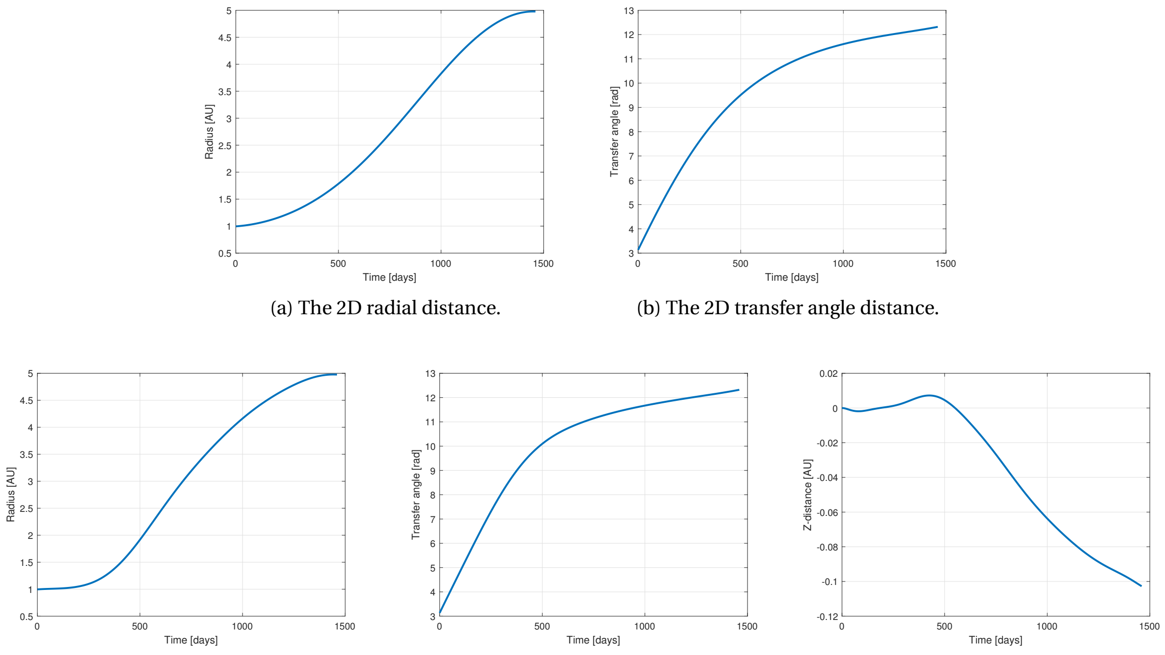
Figure 9.3: The reference trajectories to the gas giant Jupiter in two (left) and three (right) dimensions.

In Figure 9.4 the behaviour of the three components (i.e. the radial distance, the transfer angle and the axial distance) over time is shown.

The radial distance (Figures 9.4a and 9.4c) has the typical shape of a transfer orbit that comprises of only one full revolution around the Sun and could be approximated by a polynomial. It starts off with a small slope that increases rapidly to a maximum approximately halfway the trajectory after which it descends until the destination is reached. The only difference between the two- and three-dimensional curve can be seen in the first 250 days of the trajectory. The 3D approach shows a more constant behaviour during this time.

The transfer angle (Figures 9.4b and 9.4d) on the other hand, has the least interesting shape, as it is expected to constantly increase and show a logarithmic-like pattern. A trajectory would immidiately be classified as infeasible if the graph decreases, because this indicates a sudden change in direction meaning that much ΔV is unnecessary killed. The flattening pattern is caused by the increasing total distance (i.e. the root of the sum of the squares of r and z) and can be explained by Kepler's second law: The further from the Sun, the less angular distance is covered in a certain amount of time.

Finally, the axial distance (Figure 9.4e) has the most unpredictable shape of all three parameters, but it can be best described by a polynomial as well. It is largely dependent on the number of revolutions that are required to reach the target body and at what distance from the Sun this happens.



(a) The 2D radial distance.

(b) The 2D transfer angle distance.

(c) The 3D radial distance.

(d) The 3D transfer angle.

(e) The 3D axial distance.

Figure 9.4: An overview of the behaviour of the three position coordinate for the trajectories in Figure 9.3.

9.3.2. Dionysus

The second target will be the asteroid Dionysus, for its orbit adds additional complexity to the problem. It flies a fairly eccentric trajectory at an inclination that is challenging for most shape-based methods. On the other hand, the two-dimensional approximation will be less accurate. The orbital parameters of the asteroid can be seen in Table 9.4. The input parameters and outcome of the transfer trajectories are stated in Tables 9.5 and 9.6.

Due to the increased complexity in the orbit of Dionysus, it requires more ΔV to arrive there. This is mainly caused by the increased inclination, as low-thrust spacecraft are not capable of the short and optimal impulsive shots at the aphelion in order to change the orbital plane. Additionally, a large part of the ΔV is used at the end of the transfer as to match the eccentric orbit in the arrival point. The reference orbits can be found in Figure 9.5, but the directional change at the end can be best observed in Figure 9.5b.

Just as for Jupiter, the three components the trajectory to Dionysus consists of are shown in Figure 9.6. The two-dimensional radial distance (Figure 9.6a) shows one big overshoot up to a distance of 5 AU from the Sun, after which it decreases and ends up at the position of Dionysus. The large overshoot is due to the eccentric orbit of the asteroid. In combination with the current time of flight it requires to first move away from the target to eventually match its position and velocity. The three-dimensional radial distance (Figure 9.6c) how-

Table 9.4: The Keplerian elements of the asteroid Dionysus that have been used to compute an unperturbed reference orbit [22].

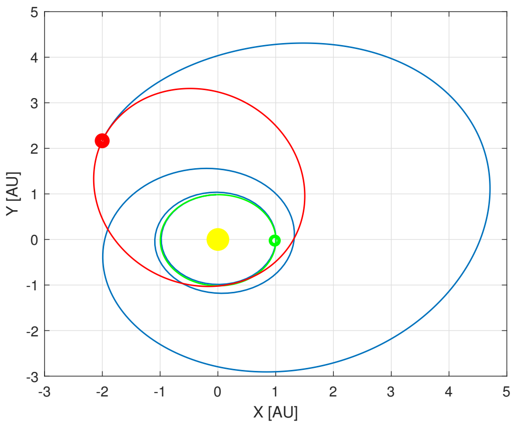
Kepler element	Value
a [AU]	2.2
e [-]	0.542
i [deg]	13.6
Ω [deg]	82.2
ω [deg]	204.2
M [deg]	114.4232
Epoch [MJD]	53400

Table 9.5: The input parameters that have been used to generate the 2D reference trajectory from Earth to Dionysus [22].

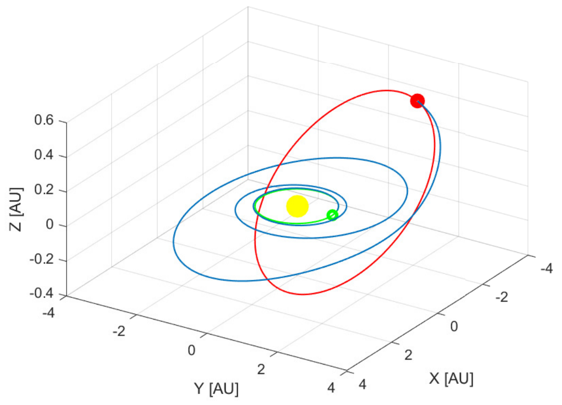
Input parameter	Value
Departure date [MJD2000]	4739.5
Time of flight [days]	3534
Discretisation points [-]	25
N_{rev} [-]	3
n_r [-]	2
n_θ [-]	5
Tolerances [-]	1×10^{-9}
Multi-start seed [-]	123
ΔV [km/s]	22.46
$T_{a_{max}}$ [mm/s^2]	0.71

Table 9.6: The input parameters that have been used to generate the 3D reference trajectory from Earth to Dionysus [22].

Input parameter	Value
Departure date [MJD2000]	4739.5
Time of flight [days]	3534
Points per revolution [-]	100
N_{rev} [-]	3
n_r [-]	6
n_θ [-]	6
n_z [-]	4
Tolerances [-]	1×10^{-9}
Multi-start seed [-]	123
ΔV [km/s]	24.05



(a) The 2D reference trajectory to Dionysus.



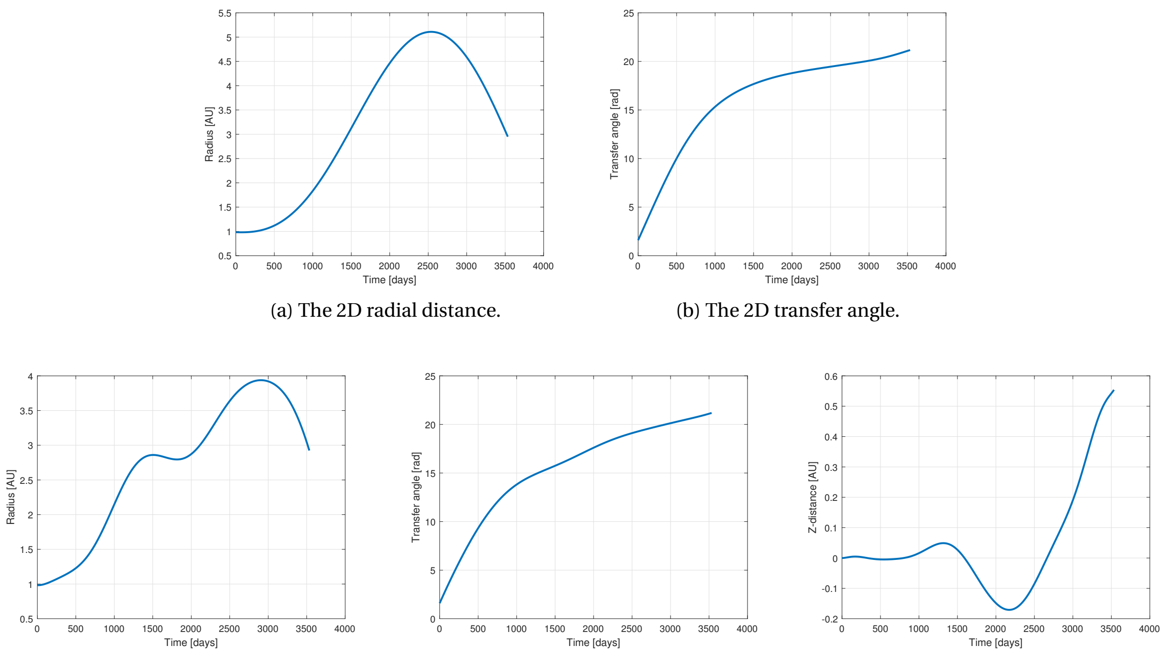
(b) The 3D reference trajectory to Dionysus.

Figure 9.5: The reference trajectories to the asteroid Dionysus in two (left) and three (right) dimensions.

ever shows two distinct bumps and has a classic polynomial shape. This can be explained by the fact that this transfer orbit first makes three full revolutions around the Sun before it reaches the asteroid. The overshoot in the last part of the trajectory is also clearly visible, as the last arc starting at about 3000 days shows a steep decrease.

The second parameter, the transfer angle (Figures 9.6b and 9.6d), again shows a logarithmic-like shape whose slope decreases over time, which is expected.

Ultimately, the z-component (Figure 9.6e) remains nearly constant during the first revolution, after which the orbital inclination changes slightly in the second revolution. During its final revolution it is at such a distance from the Sun that it takes nearly half the time of flight of the entire trajectory to arrive at the target destination.



(a) The 2D radial distance.
(b) The 2D transfer angle.
(c) The 3D radial distance.
(d) The 3D transfer angle.
(e) The 3D axial distance.

Figure 9.6: An overview of the behaviour of the position components for the trajectory in Figure 9.5.

9.3.3. Haumea

Haumea is a particularly interesting research object, as it is one of the few known trans-Neptunian dwarf planets, of which only four are known. Besides, it was one of the first objects of its size that posesses a ring system [3]. The reason it has been considered as a test object, is because it basically combines the properties of the two previous targets: both the semi-major axis, the inclination, and the eccentricity are a great challenge for a trajectory computed by a finite Fourier series. It should however be noted that due to the high inclination, there is no point in applying the two-dimensional finite Fourier series method to this problem, as the validity of this approach only holds up to inclinations of approximately 15° [21]. The parameters and outcome of the transfer trajectory can be seen in Table 9.7, while the orbital parameters that have been used to reconstruct the unperturbed orbit of Haumea can be found in Table 9.8.

Table 9.7: The input parameters that have been used to generate the reference trajectory from Earth to Haumea [18].

Input parameter	Value
Departure date [MJD2000]	12422.5
Time of flight [years]	70
Points per revolution [-]	500
N_{rev} [-]	14
n_r [-]	6
n_θ [-]	6
n_z [-]	8
Tolerances [-]	1×10^{-9}
ΔV [km/s]	161.6

Table 9.8: The Keplerian elements of the dwarf planet Haumea that have been used to compute an unperturbed reference orbit [9].

Kepler element	Value
a [AU]	43.1819
e [-]	0.19489
i [deg]	28.2135
Ω [deg]	122.1627
ω [deg]	238.77988
M [deg]	217.7719
Epoch [MJD]	59000

Unfortunately, as is evident from Figure 9.7, the method is not capable of finding a suitable trajectory at all. The z-component of the trajectory immediately goes down nearly 30 AU, after which it recovers and still reaches its final destination. Even with this combination of the departure date and the time of flight, the axial coordinate of Haumea is only 1.97 AU lower than the Earth, which is in the same range as the asteroid Dionysus for which a feasible trajectory could be found. Nonetheless, this shape and the corresponding

ΔV deem this solution to be infeasible. Therefore it was decided to exclude Haumea from the initialisation analysis.

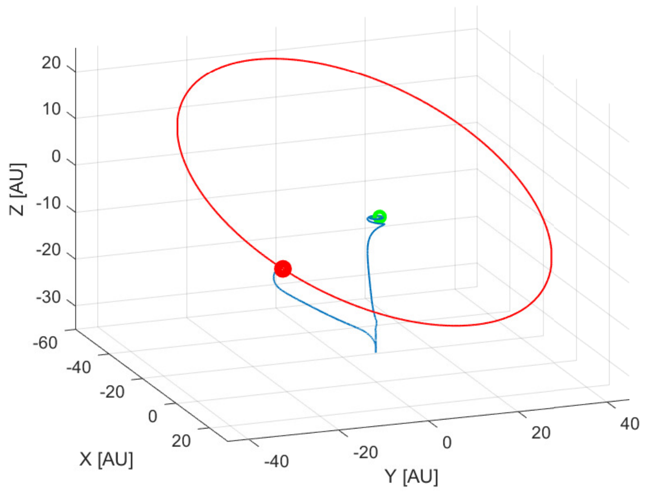


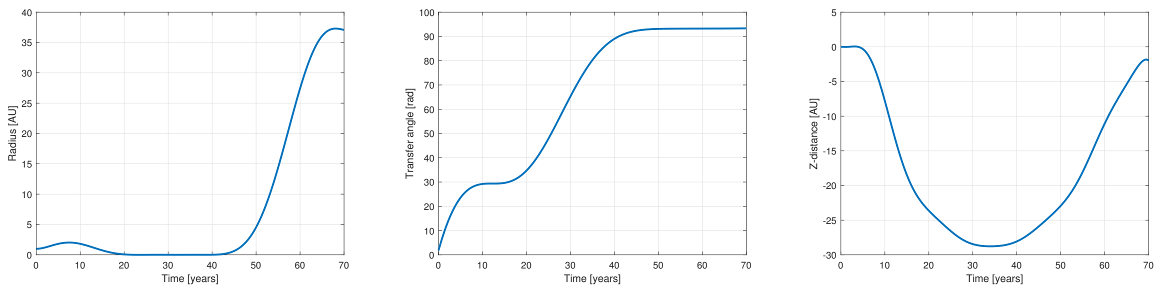
Figure 9.7: The reference trajectory to Haumea.

Even though the trajectory does not make sense at all, it is still worth to take a look at the development of the three trajectory components in Figure 9.8.

At first sight, the radial distance (Figure 9.8a) confirms the flaws of this transfer orbit. It starts in a normal way, but after about 20 years it remains constant at a level of nearly zero. This region obviously depicts the downwards movement as seen in Figure 9.7. The final increase in radius is caused by the uprising from the lowest part of the descent to the location of the dwarf planet.

Furthermore, the transfer angle (Figure 9.8b) looks quite normal. Its overall shape is in line with a feasible trajectory solution, however the small constant region between 10 and 20 years is unexpected.

Lastly, the axial distance (Figure 9.8c) depicts a large valley, which can be fully contributed to the unnatural downward movement of this trajectory solution.



(a) The radial distance. (b) The transfer angle. (c) The axial distance.

Figure 9.8: An overview of the behaviour of the three position components for the trajectory in Figure 9.7.

9.4. Experimental Outline

Now the different functions have been explained, the test objects have been defined and the reference trajectories have been generated, the research procedure can be formulated. As mentioned in Section 9.1, the goal is to investigate whether a more appropriate initialisation function exists for a certain trajectory design problem that is capable of passing an initial guess to the solver such that the overall computation time is shorter than it is now and that the method is more stable in three dimensions. Note that all test runs will be performed on an Intel Core i7-4700MQ 2.40 GHz with 8.00 GB RAM on Windows 10.

9.4.1. 2D Test Set-Up

I. The first two-dimensional experiment will simply replace the standard initialisation function as described in Section 9.2.1 and use one of the three approaches that were spelled out in Sections 9.2.2 to 9.2.4 to estimate r and θ.

II. The second analysis uses a more tailored focus. Based on the natural behaviour of the different components, as displayed in Figures 9.4 and 9.6 the most appropriate initialisation function is chosen. In short, this entails that the radial component is approximated by a polynomial function, while the transfer angle is estimated by means of a logarithmic function.

9.4.2. 3D Test Set-Up

I. The first experiment in 3D will assess the convergence speed of the finite Fourier series solver when an initialisation function is used other than the power function. This means that a trajectory to Jupiter and Dionysus is computed where the initial guess for the coefficients in r and θ is found by the three functions described in Section 9.2. In this experiment the coefficients for the $z-$coordinate will still be set to zero, as to resemble the procedure by Taheri and Abdelkhalik [22].

II. The second experiment will basically have the same set-up as the first one, however, this time the axial coordinate is initialised by the functions from Section 9.2 as well.

III. The final experiment will take a more in-depth look at the independent coordinates the trajectory is built up from. Based on Figures 9.4 and 9.6, it was decided that r will be approximated by a power function, θ will be approximated by a logarithmic function and z will either be set to zero, or it will be approxiamted by a sinusoidal function.

9.4.3. Time Assessment

For each case, a comparison with respect to the reference trajectories will be made in terms of computation time and the computed ΔV value. The ΔV merely functions as a control variable to assess the quality of the newly obtained solution. In case one of the proposed initialisation methods has a feasible ΔV outcome, an assessment of the computation time is done. To do so, a clear distinction between the two steps in the solving procedure needs to be made. First, the initialisation time will be measured, after which the actual solver time will be determined.

The initialisation time is the time required to compute the coefficients of the initialisation functions in all dimensions, fill the matrices with the Fourier approximations, and solve for the approximation that results in the initial guess for the main Fourier solver in the algorithm.

Later on, the main solving time is measured. As soon as the initial guess is passed to the solver, the Nelder-Mead algorithm is activated and starts moving towards the right set of Fourier coefficients that yield a feasible solution.

In order to diminish the effect of background processes on the computer, all other programmes were closed. However, it is always possible that the computer suddenly starts to do some housekeeping tasks. Therefore 1000 identical runs have been performed and the mean of these activities is used as a solid measure regarding the required total solving time.

In case the CPU-time measurements still show a large spread, an outlier analysis will be done. This will flatten the peaks and prevents the average CPU-time from being determined by a few data points that are actually far off.

10
2D Results

As described in the previous chapter, the two-dimensional analysis comprises of two parts. First, the effects of a different initialisation function applied to both the radial and the transverse component will be demonstrated in Section 10.1. Consecutively, the outcome of the case for which tailored initialisation functions have been used for each component is displayed in Section 10.2. Recall that the input and reference parameters of the trajectories to Jupiter and Dionysus can be found in Tables 9.1 and 9.2, and Tables 9.4 and 9.5 respectively. The chapter ends with a concise conclusion of the obtained results in Section 10.3.

10.1. Using a Different Initialisation Function

The first case of the two-dimensional initialisation analysis comprises of testing three new functions to generate a priori values to steer the solver into the right direction. With this approach, both the radial distance and the transfer angle will be approached by either a power function, an exponential function, a sinusoidal function and a logarithmic function. First the results for the Jupiter test case will be treated, after which the Dionysus test case will be explained.

10.1.1. Jupiter

The results of the four different approximation functions are shown in Table 10.1. With the power function initialisation operating as a reference value, it is interesting to note that another solution with an even lower ΔV outcome has been found when the solver is initialised by a sinusoidal function. The average of the ΔVs is 18.22 km/s and the spread σ is only 0.085 km/s.

Table 10.1: The outcome of the solver when the a priori coefficients are computed by different functions that were applied to generate an a priori estimate of the r- and θ-components for an unconstrained trajectory to Jupiter.

Function	ΔV [km/s]
Power	18.22
Exponential	18.30
Sinusoidal	18.10
Logarithmic	18.25

The four outcomes are shown in Figure 10.1, but as the outcomes are very similar, it is hard to spot a difference between the four figures. An exception can be observed for the sinusoidal approximation, for which the minimum orbital distance passes 0.06 AU more in the negative y-direction. This is indicated by the blue line, but still it is hardly notable in the figure.

Because all four initialisation functions yielded feasible results, a CPU time test has been set up in order to measure the computational speed. The results of both CPU time tests have been collected in Table 10.2. Note that the two processes are expressed in a different unit. From here it becomes clear that the logarithmic func-

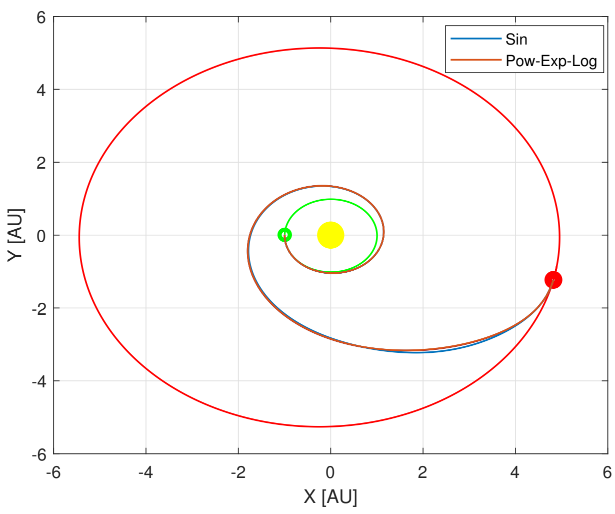


Figure 10.1: The trajectories to Jupiter computed with four different initialisation functions that were applied to generate an a priori estimate of the r- and θ-components.

tion requires the least time for the initialisation process, while the exponential approximation is the fastest in terms of the main solving process: a gain of about 10%.

Table 10.2: An overview of the time required to compute the coefficients of the initialisation function and the actual Fourier coefficients for a single-revolution trajectory to Jupiter.

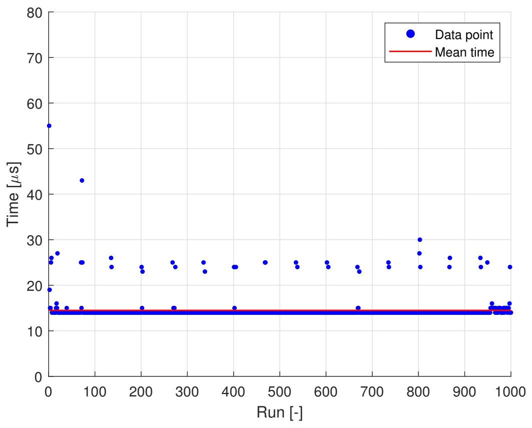
Function	Initialisation time [μs]	Solver time [ms]
Power	14.5	312.0
Exponential	15.1	287.2
Sinusoidal	23.0	419.5
Logarithmic	13.5	341.0

The results from the initialisation process are displayed in Figure 10.2. All four graphs show some outliers, but most data points are on one line, which is expected. Because the time unit is only microseconds, only 0.015 seconds have passed over the entire run of these simulations. This is such a short time span that it is nearly impossible for the computer to initiate a computationally intense background process.

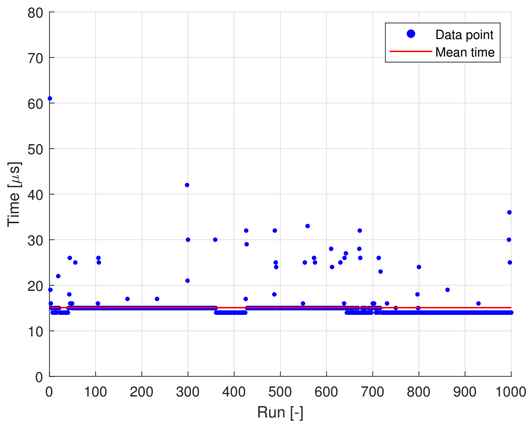
More valuable results are shown in Figure 10.3, as this figure contains the runtime of the actual Fourier coefficients solver, which is a more complex procedure than the initialisation process. The same principle that explains the nearly absent spread in Figure 10.2 shows the exact opposite in Figure 10.3. The average solver time with a priori values that have been generated by an exponential approximation, for example, is about 0.3 ms, which brings the total time required to complete all 1000 simulations to approximately five minutes. These five minutes span an interval during which it is highly likely that some background processes are conducted by the computer. Figures 10.3c and 10.3d showed many bumps that greatly influenced the average CPU-time. Therefore an outlier analysis has been done for these two initialisation approaches has been done. The original uncorrected figures can be seen in Appendix G.

10.1.2. Dionysus

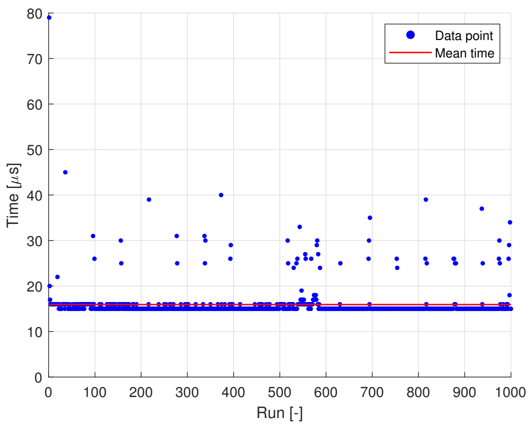
In contrast to the outcome of the test trajectory to Jupiter, the trajectory to Dionysus does provide a more constant range of results. In Table 10.3 the effect of the four different initialisation functions in terms of required ΔV can be seen. All three new initialisation functions manage to find the same result as the original power function. The puny difference can be attributed to rounded values.



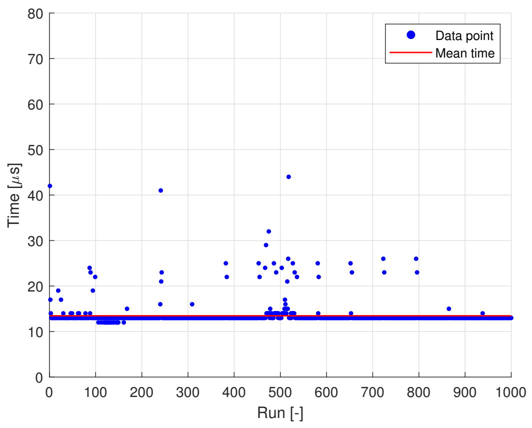
(a) $f(t) = at^3 + bt^2 + ct + d$



(b) $f(t) = ae^{bt} + c$



(c) $f(t) = A\sin\left(\omega\left(t-\phi\right)\right) + K$



(d) $f(t) = a + b\ln(t)$

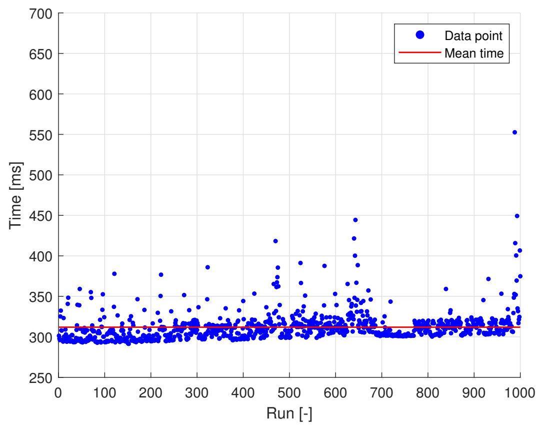
Figure 10.2: An overview of the average CPU-time required to find the coefficients of the initialisation function for a trajectory to Jupiter.

Table 10.3: The outcome of the solver when the a priori coefficients are computed by different functions that were applied to generate an a priori estimate of the r- and θ-components for an unconstrained trajectory to Dionysus.

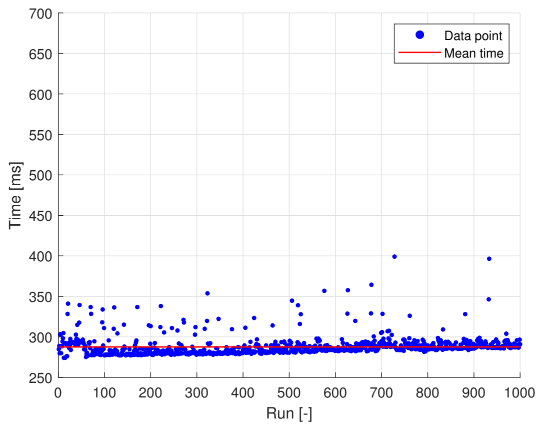
Function	ΔV [km/s]
Power	22.46
Exponential	22.46
Sinusoidal	22.46
Logarithmic	22.47

As the three new initialisation functions yielded the exact same outcome, their trajectories are identical as well. The triple-revolution trajectory is shown in Figure 10.4.

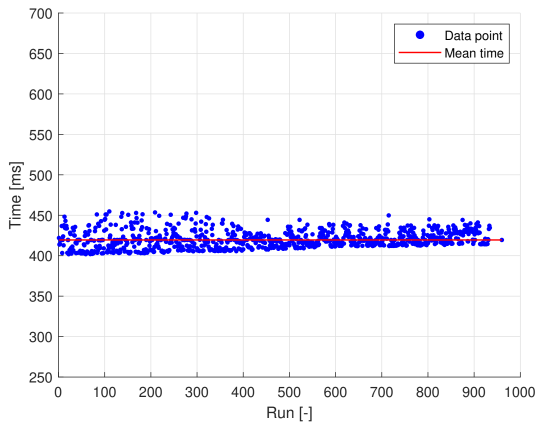
The necessary computation times however do vary much from each other. Both for the initialisation time and the main solver time large varieties are observed. The obtained results are presented in Table 10.4. In here it can be seen that the logarithmic function is the fastest in terms of initialisation, while an exponential approximation provides the fastest converging time for the main Fourier coefficient solver. It is especially remarkable that the required time is nearly 50% faster than the reference power function while it also manages to retain the optimum.



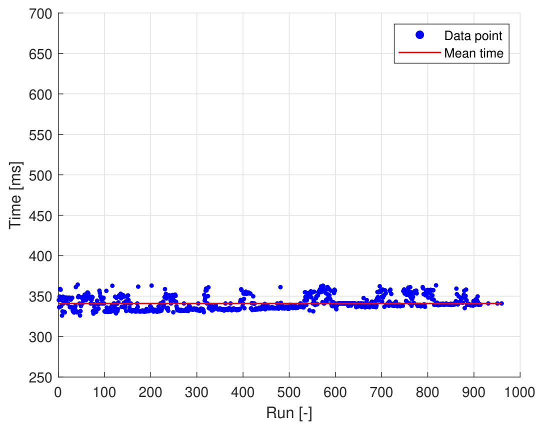
(a) $f(t) = at^3 + bt^2 + ct + d$



(b) $f(t) = ae^{bt} + c$



(c) $f(t) = A\sin\left(\omega\left(t - \phi\right)\right) + K$



(d) $f(t) = a + b\ln(t)$

Figure 10.3: An overview of the average CPU-time required to find the Fourier coefficients that shape the trajectory to Jupiter.

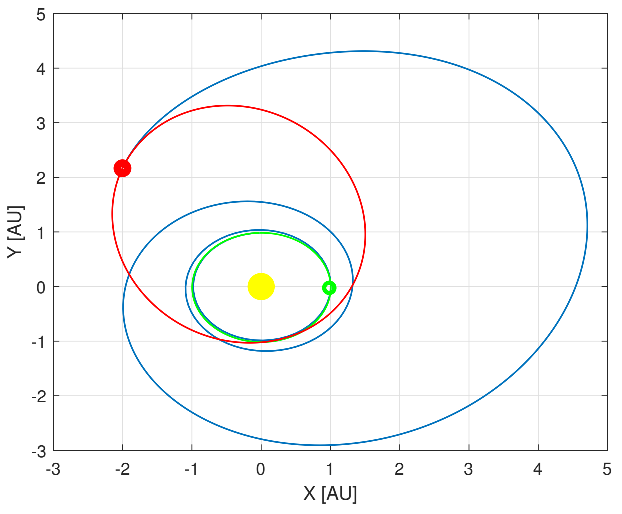
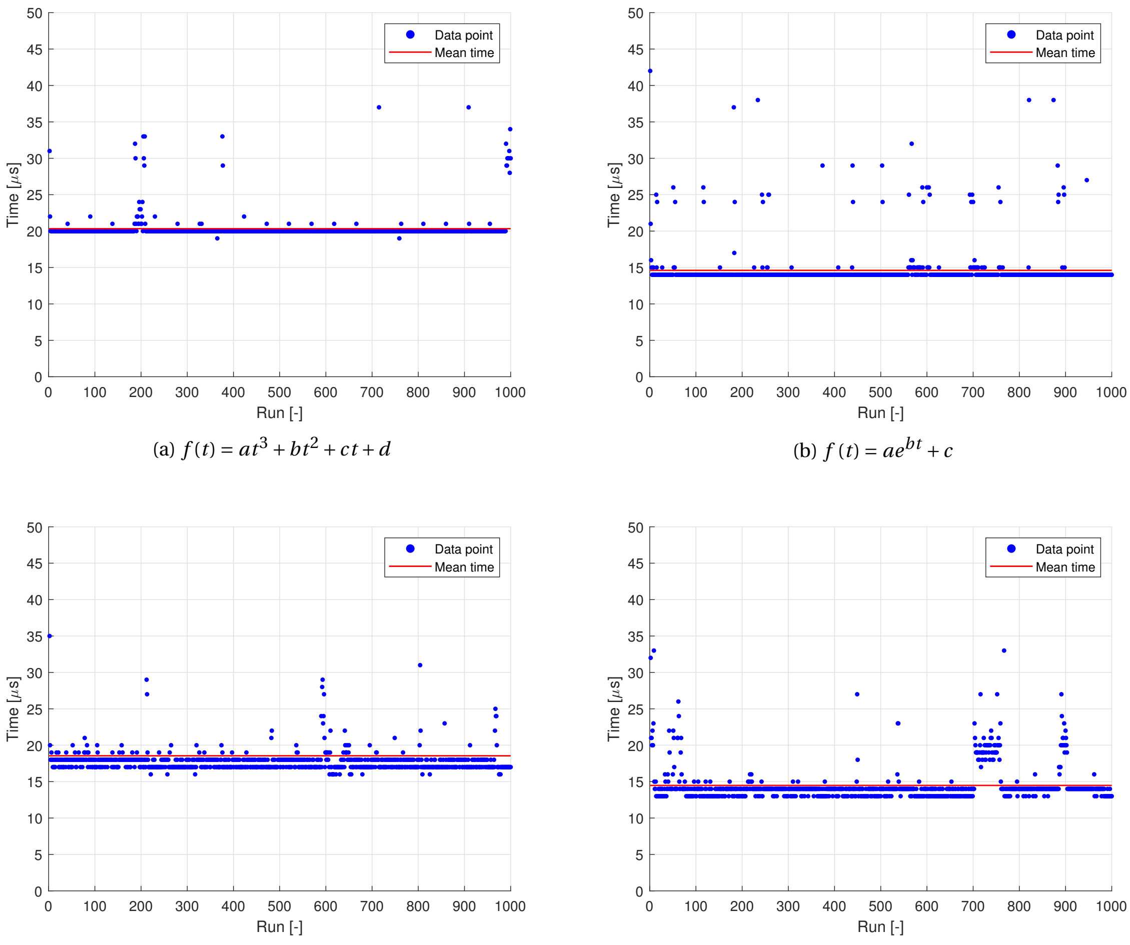


Figure 10.4: The trajectories to Dionysus computed with four different initialisation functions that were applied to generate an a priori estimate of the r- and θ-components. All four functions generated a nearly identical trajectory.

Table 10.4: An overview of the time required to compute the coefficients of the initialisation function and the actual Fourier coefficients for a triple-revolution trajectory to Dionysus.

Function	Initialisation time [µs]	Solver time [ms]
Power	20.3	278.4
Exponential	14.6	159.8
Sinusoidal	21.9	209.3
Logarithmic	14.5	273.9

The initialisation time graphs in Figure 10.5 again show nothing astounding, as the same explanation as for Jupiter applies: in such a short time span it is barely impossible to initiate large and complex processes for a computer.



(a) $f(t) = at^3 + bt^2 + ct + d$

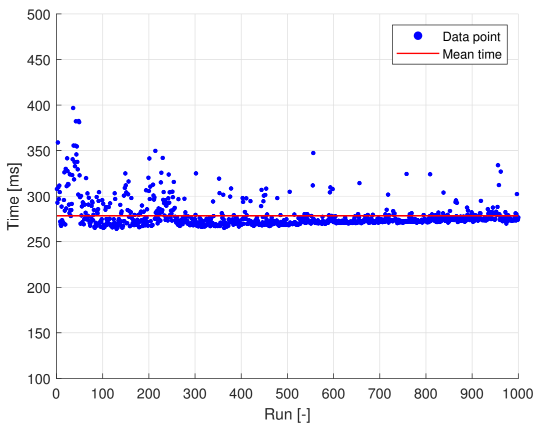
(b) $f(t) = ae^{bt} + c$

(c) $f(t) = A\sin\left(\omega\left(t - \phi\right)\right) + K$

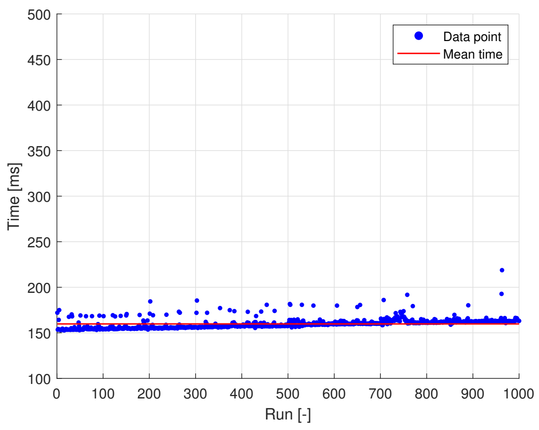
(d) $f(t) = a + b\ln(t)$

Figure 10.5: An overview of the average CPU-time required to find the coefficients of the initialisation function for a trajectory to Dionysus.

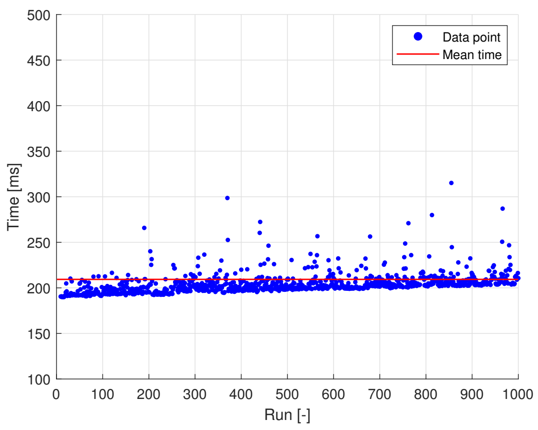
The convergence time of the Fourier coefficient solver in Figure 10.6 shows one clear exception: the exponential function in Figure 10.6b has a significantly better converging time than the other three techniques.



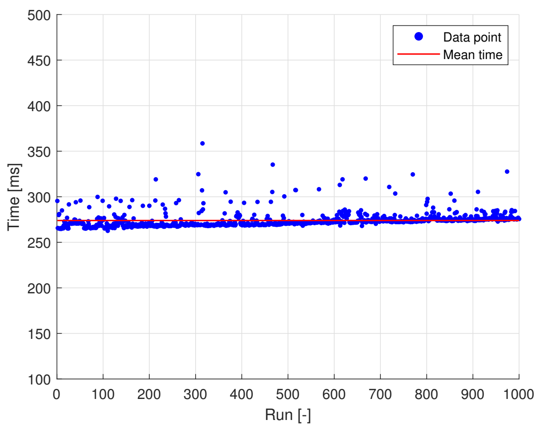
(a) $f(t) = at^3 + bt^2 + ct + d$



(b) $f(t) = ae^{bt} + c$



(c) $f(t) = A\sin\left(\omega\left(t-\phi\right)\right) + K$



(d) $f(t) = a + b\ln(t)$

Figure 10.6: An overview of the average CPU-time required to find the Fourier coefficients that shape the trajectory to Dionysus.

10.2. Using a Tailored Initialisation Function

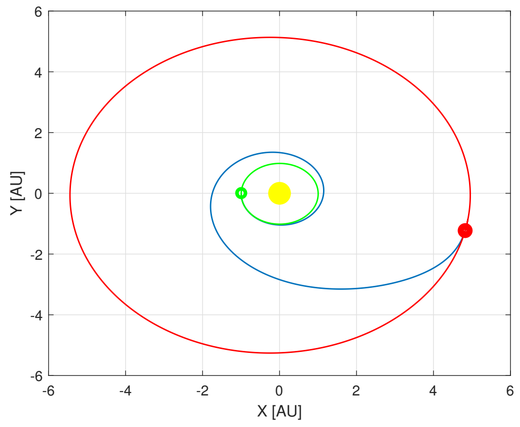
As a second experiment, it has been investigated if the convergence speed of the algorithm can be increased if tailored initialisation functions are used to approximate the Fourier coefficients. Based on Figures 9.4 and 9.6, the radial components are still modelled by a power function, but the transfer angle usually has the shape of a logarithmic function. The results from this analysis are displayed in Table 10.5. For the test case targeting Jupiter a feasible solution has been found, but it shows a higher ΔV than was shown in Section 10.1. The test trajectory to Dionysus did not yield a feasible solution, as a ΔV of nearly 100 km/s is not attainable. Therefore it has been decided to omit the initialisation and solver convergence time analysis, as there is no point in measuring the computation time of an invalid solution.

Table 10.5: The outcome and convergence time of the solver when the a priori values are computed by approximating r by a polynomial function and θ by a logarithmic function.

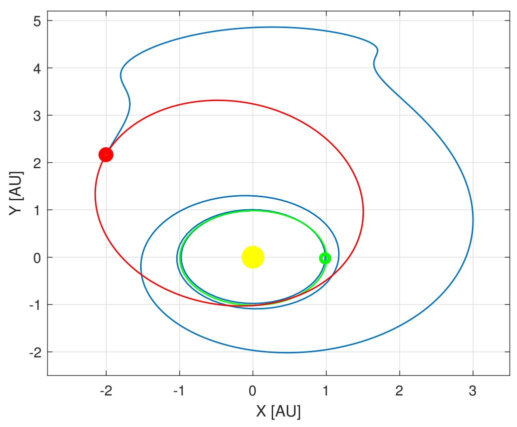
Function	Component	Function	ΔV [km/s]	Initialisation time [µs]	Solver time [ms]
Jupiter	radial transverse	Power Logarithmic	19.0	19.6	331.1
Dionysus	radial transverse	Power Logarithmic	99.60	-	-

The two trajectories are shown in Figure 10.7. The trajectory to Jupiter (Figure 10.7a) hardly differs from the one in Figure 10.1, but some more thrust must have been applied in order to end up at a ΔV which is 1 km/s higher than the known best value of Table 10.1.

The transfer orbit to Dionysus (Figure 10.7b) however immediately explains the high ΔV values. The trajectory contains two large kinks during which much more thrust is required to suddenly move into a direction other than the expected circular path, which leads to the high ΔV value.



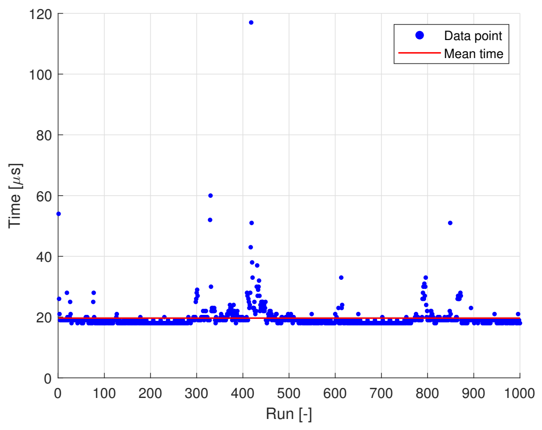
(a) The trajectory to Jupiter where r is approximated by a power function and θ by a logarithmic function.



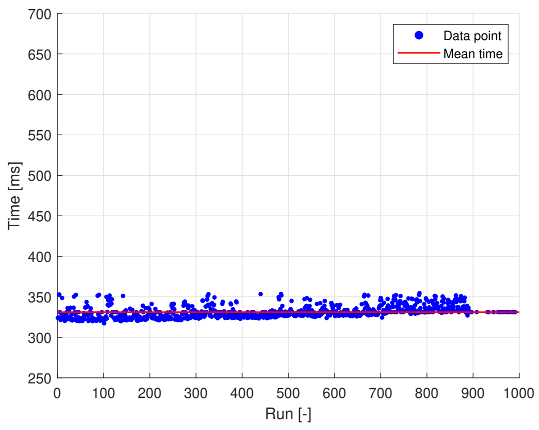
(b) The trajectory to Dionysus where r is approximated by a power function and θ by a logarithmic function.

Figure 10.7: The trajectories to Jupiter (left) and Dionysus (right) where r and θ both have been approximated by a different function.

The time examination of the trajectory to Jupiter is depicted in Figure 10.8b. Note that this figure has been corrected for its high number of outliers. The original uncorrected figure can be observed in Appendix G. When Figure 10.8b is compared to the initialisation and solver convergence time in Figures 10.2 and 10.3 it is easily concluded that the method is not capable of surpassing the results from the first experiment. It is both slower regarding initialisation and convergence speed. Besides, the achieved ΔV value is not more beneficial either.



(a) The average CPU-time required to solve for the coefficients of the initialisation function.



(b) The average CPU-time required to solve for the Fourier coefficients.

Figure 10.8: The initialisation time (left) and the solver convergence time (right) of the trajectory to Jupiter that uses a polynomial function to approximate r, while a logarithmic function has been used to estimate θ.

10.3. Conclusion

The previously elaborated data leads to the conclusion that there exists an initialisation function that generates better a priori values for the two-dimensional version of the finite Fourier series method such that the correct solution is found in less time than the conventional way by Taheri and Abdelkhalik [21]. For both the trajectory to Jupiter and Dionysus it is the exponential function that is the fastest. A transfer orbit to Jupiter is computed 7.9% faster, while the trajectory to Dionysus is computed 42.6% faster.

11
3D Results

The results of the three-dimensional analysis as introduced in Section 9.4 are presented here. The three trajectory case studies are found in Sections 11.1 to 11.3. After that, an interpretation of the impact of the different initialisation functions on the behaviour of the Fourier coefficients in particular is given in Section 11.4. The chapter is concluded with an explanation on the effects of a multi-start approach in Section 11.5. It should be noted that the input and reference parameters of the trajectories to Jupiter and Dionysus can be found in Tables 9.1 and 9.3, and Tables 9.4 and 9.6 respectively.

Note that the multi-start procedure is only activated if the first solution obtained from the initial guess is infeasible. A maximum of 15 runs will be done. The multi-start values that are mentioned in the tables below indicate the best solution obtained from those 15 runs. However, in case the initial guess that is generated by the initialisation function provides a feasible solution, the cell is left empty.

11.1. Using an Alternative Initialisation Function (r and θ Only)

To recap Section 9.4, the first case comprises of using alternative initialisation functions for the Fourier coefficients that describe the behaviour of r and θ, while the Fourier coefficients for the z coordinate are set equal to zero. First the results from the trajectory to Jupiter will be presented, after which the Dionysus test case is described.

11.1.1. Jupiter

Table 11.1 contains the results of the four different initialisation function runs. The trajectory that has been generated with a power function approximation serves as a reference. From the table it becomes clear that it is only the exponential approximation that initialises the solver such that it finds a feasible solution.

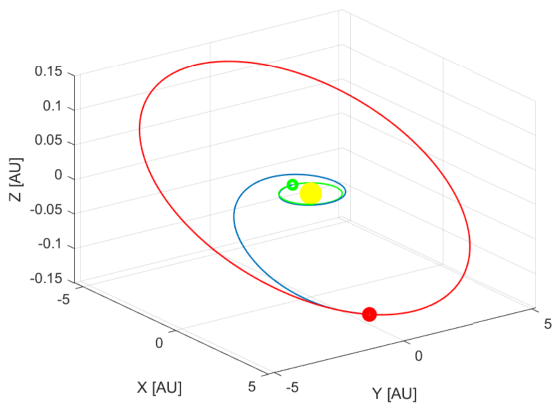
Table 11.1: The outcome of the solver when the a priori coefficients are computed by different functions that were applied to the r- and θ-components for a trajectory to Jupiter.

Function	ΔV [km/s]	Best Multi-start ΔV [km/s]
Power	16.74	-
Exponential	18.59	-
Sinusoidal	127.64	155.62
Logarithmic	206.92	8446.36

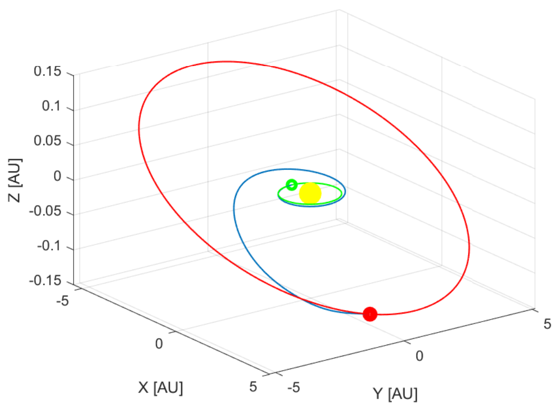
The trajectories corresponding to the solutions from Table 11.1 are shown in Figure 11.1. The reference trajectory can be seen in Figure 11.1a in the top left corner. Observing the four figures, some interesting remarks can be made.

The second best result, generated by an exponential initialisation function, depicted in Figure 11.1b shows a very similar trajectory as in Figure 11.1a. The slight ΔV difference of about 2 km/s can be explained by the slight overshoot of Jupiter's orbit and the stronger inclination change at the start of the transfer. The overshoot has to be corrected for at the end, hence built-up speed is killed and an inclination change is more efficient when it is done at a larger distance, as the orbital speed will be lower.

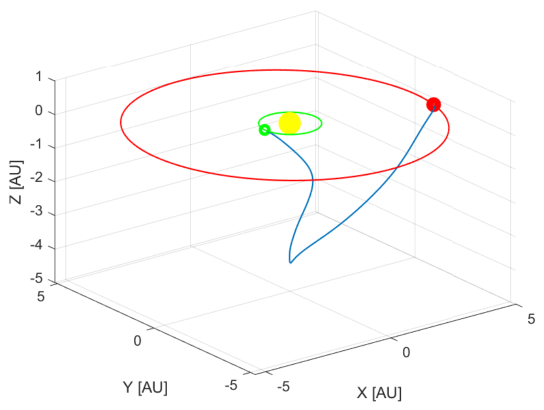
Furthermore, the sinusoidal and logarithmic functions are not capable of producing a feasible trajectory solution. This is especially surprising for the sinusoidal case, as a Fourier series actually consists of a summation of this type of functions. The exceptionally high ΔV values are explained by the steep movement in the negative z-direction, after which all velocity needs to be directed the exact opposite way.



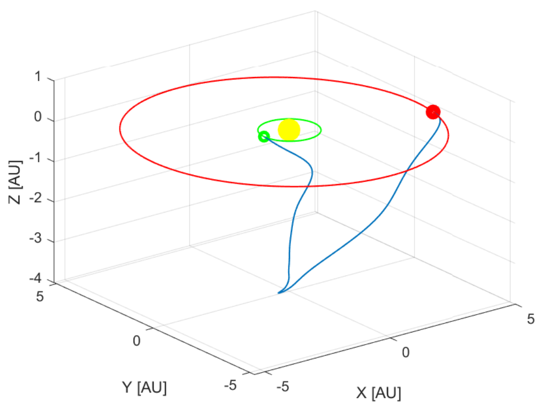
(a) $f(\tau) = a\tau^3 + b\tau^2 + c\tau + d$



(b) $f(\tau) = ae^{b\tau} + c$



(c) $f(\tau) = A\sin\left(\omega\left(\tau - \phi\right)\right) + K$



(d) $f(\tau) = a + b\ln(\tau)$

Figure 11.1: The trajectories to Jupiter computed with four different initialisation functions that were applied to generate an a priori estimate of the r- and θ-components.

11.1.2. Dionysus

The results of the different initialisation strategies for a more complex transfer orbit due to its higher inclination and eccentricity are shown in Table 11.2. In contrast to the trajectory to Jupiter, it is only the original power initialisation function that provides a feasible solution. The other three functions are not able to guide the solver towards a reasonable solution. Even with a multi-start procedure to increase the chances of finding this solution, it fails to converge. Especially the logarithmic trajectory approximation has a strong tendency to blow up.

The actual trajectories that correspond to the solutions presented in Table 11.2 are depicted in Figure 11.2. Again, the reference trajectory that was generated with a power function approximation is shown in the top left corner of the figure (Figure 11.2a). A quick look at the three other trajectory solutions explains the peculiar ΔV results.

Table 11.2: The outcome of the solver when the a priori coefficients are computed by different functions that were applied to the r- and θ-components for a trajectory to Dionysus.

Function	ΔV [km/s]	Best Multi-start ΔV [km/s]
Power	24.05	-
Exponential	124.49	1140.52
Sinusoidal	169.04	1800.64
Logarithmic	2158.88	1.0876×10^6

The exponential and sinusoidal approximation in Figures 11.2b and 11.2c stay within the radial distance bounds in the xy-plane, but tend to overshoot in the axial direction. Moreover, both trajectories show the same pattern of flying into the opposite direction at departure, which asks for large quantities of ΔV to do so and later on in the trajectory, they contain another sudden change of direction adding up to the total ΔV budget.

Finally, the logarithmic approximation in Figure 11.2d provides the least useful, but most spectacular result. Its spatial path contains multiple changes in direction, it severely exceeds the radial distance as measured from the Sun in the xy-plane and it has a tendency for diving down nearly 15 AU. The fact that the time of flight for all four trajectories is the same explains why a significantly higher overall velocity is needed to actually fly such extreme distances. The combination with the many, sudden directional changes leads to a total ΔV of 2158.88 km/s.

11.2. Using a Different Initialisation Function (r, θ and z)

For the second case set-up, a closer look is taken at the Achilles' heel of most shape-based methods: the 3D component, which is in this case represented by the z-coordinate due to the choice for the cylindrical coordinate system. In this study case, it is not only the r- and θ-coordinate that are initialised by a different function other than a polynomial, but also the z-component. In the original method by Taheri and Abdelkhalik [22] this is always set to zero. First, the results for the transfer trajectory to Jupiter will be explained, after which the results for Dionysus are presented.

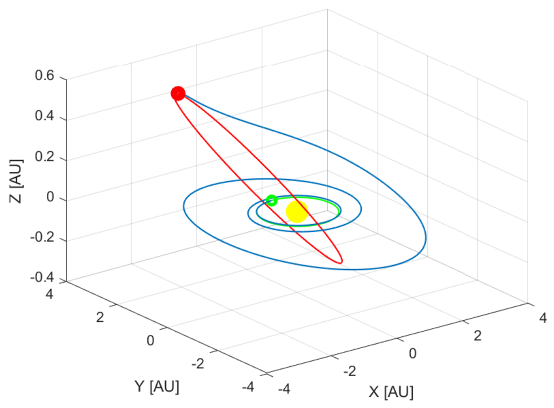
11.2.1. Jupiter

Just as for the previous case explained in Section 11.1, it is again the original power function and the exponential function approximation that allow the solver to converge to a feasible solution, as can be seen in Table 11.3. However, if these two results are compared to Table 11.1, it stands out that even these first two runs are not the same as in the original definition of the finite Fourier series method. This proves that this initialisation method too is very sensitive to a priori information when it comes to the three-dimensional component. It goes without saying that the sinusoidal and logarithmic initialisation functions yield a worthless outcome.

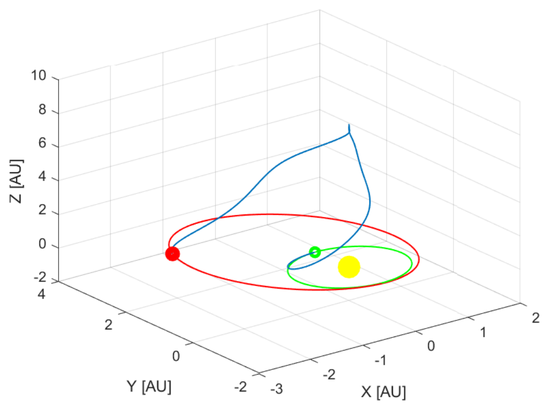
Table 11.3: The outcome of the solver when the a priori coefficients are computed by different functions that were applied to the r-, θ and z-components for a trajectory to Jupiter.

Function	ΔV [km/s]	Best Multi-start ΔV [km/s]
Power	19.29	-
Exponential	23.69	-
Sinusoidal	141.24	182.29
Logarithmic	424.31	21657.2

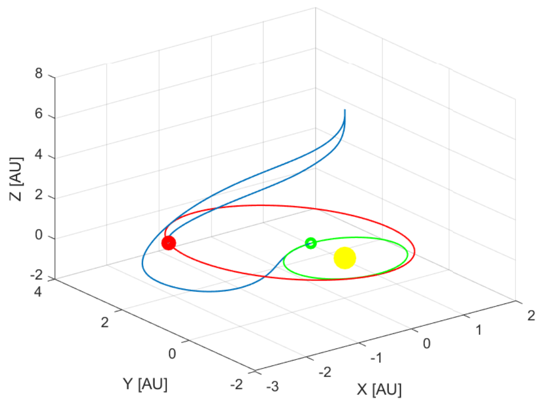
The four orbits are shown in Figure 11.3. The difference in the required ΔV for the power and exponential function (Figures 11.3a and 11.3b) can be attributed to the strong tendency of the exponential approximation to increase the inclination of the transfer orbit at a very early point along the path, which could be seen in Section 11.1 as well. If this would be done at larger distance from the Sun, the orbital speed is lower, hence the amount of ΔV to do such an inclination change is less.



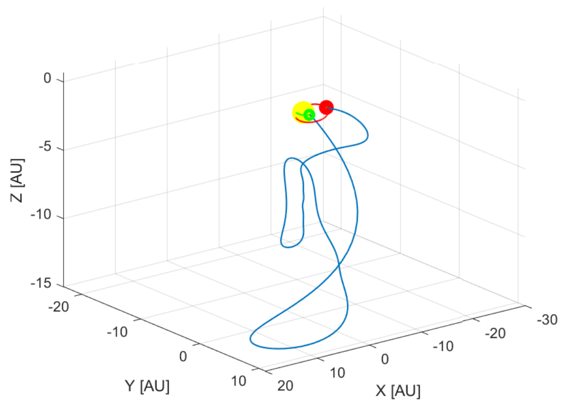
(a) $f(\tau) = a\tau^3 + b\tau^2 + c\tau + d$



(b) $f(\tau) = ae^{b\tau} + c$



(c) $f(\tau) = A\sin\left(\omega\left(\tau - \phi\right)\right) + K$



(d) $f(\tau) = a + b\ln(\tau)$

Figure 11.2: The trajectories to Dionysus computed with four different initialisation functions that were applied to generate an a priori estimate of the r- and θ-components.

The sinusoidal and logarithmic trajectories in Figures 11.3c and 11.3d obviously explain the high ΔV values. A steep downward movement and (multiple) abrupt changes in direction result in infeasible ΔV values of 141.24 and 424.31 km/s, respectively.

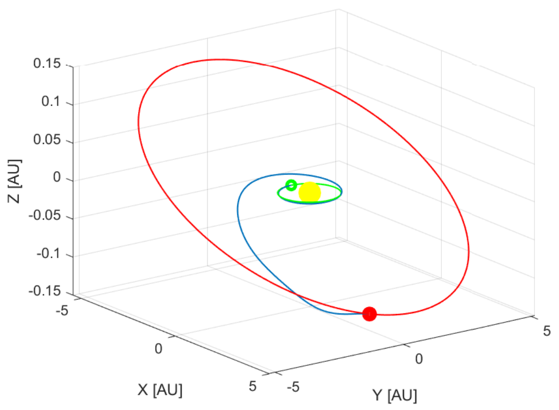
11.2.2. Dionysus

When the same procedure is applied to the asteroid Dionysus, not a single feasible trajectory is found. The outcome of the extended initialisation analysis for the three-dimensional component is displayed in Table 11.4. Besides the fact that all functions fail to converge to an appropriate solution, another interesting observation is made in terms of the *best* performer. It is not expected that the exponential approximation manages to find a lower total ΔV than the power series, which provided the best solution in all previously described cases.

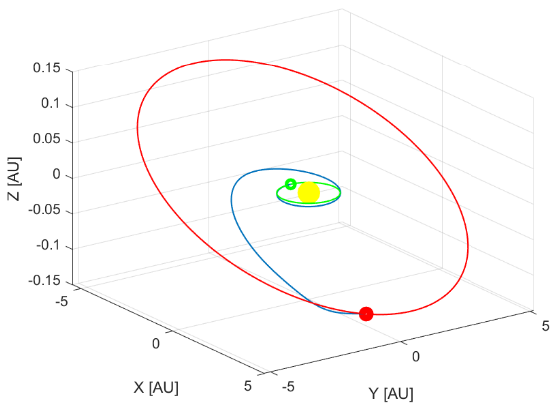
In Figure 11.4 the most improbable transfer orbits are shown. All four show an interesting shape, but unfortunately all of them are impossible to fly, rendering them infeasible.

11.3. Tailored Initialisation Functions

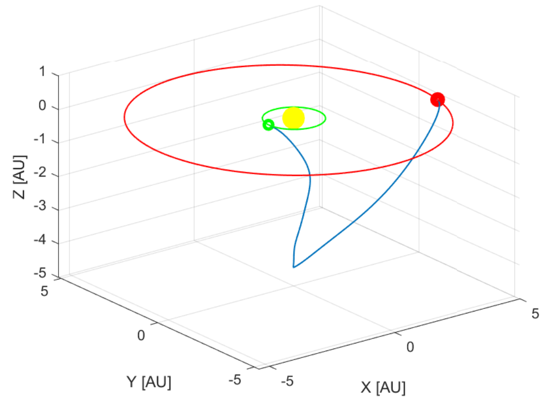
The final study case makes use of the most appropriate initialisation function depending on the natural behaviour of the three position coordinates. This entails that for the initial guess of the coefficients for each trajectory coordinate a different function can be applied. Furthermore, due to the highly-sensitive behaviour of the z-component, this initialisation set-up is also used to only approximate r and θ, whilst z is set equal



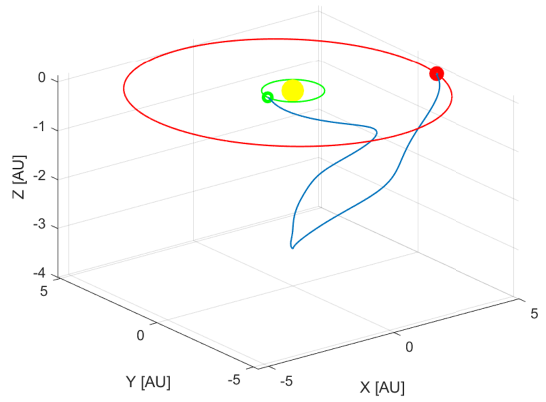
(a) $f(\tau) = a\tau^3 + b\tau^2 + c\tau + d$



(b) $f(\tau) = ae^{b\tau} + c$



(c) $f(\tau) = A\sin\left(\omega\left(\tau - \phi\right)\right) + K$



(d) $f(\tau) = a + b\ln(\tau)$

Figure 11.3: The trajectories to Jupiter computed with four different initialisation functions that were applied to generate an a priori estimate of the r-, θ and z-components.

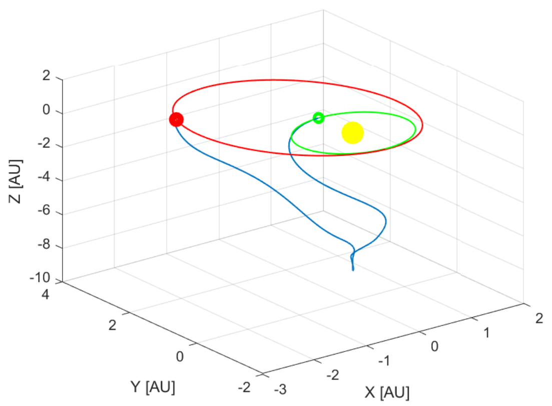
Table 11.4: The outcome of the solver when the a priori coefficients are computed by different functions that were applied to the r-, θ and z-components for a trajectory to Dionysus.

Function	ΔV [km/s]	Best Multi-start ΔV [km/s]
Power	134.73	12362.4
Exponential	125.52	692.31
Sinusoidal	144.64	30142.2
Logarithmic	1806.83	1.1085×10^6

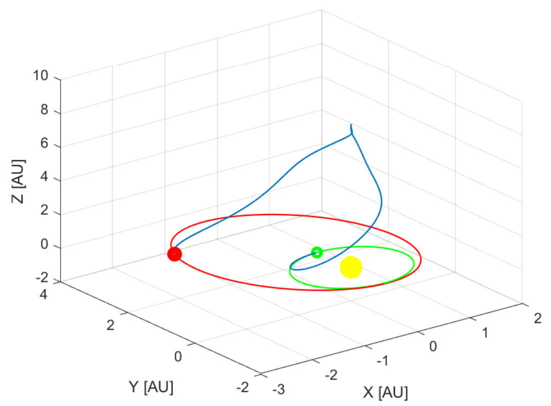
to zero. The results for the transfer orbit to Jupiter will be demonstrated first, after which the transfer orbit to Dionysus are presented.

11.3.1. Jupiter

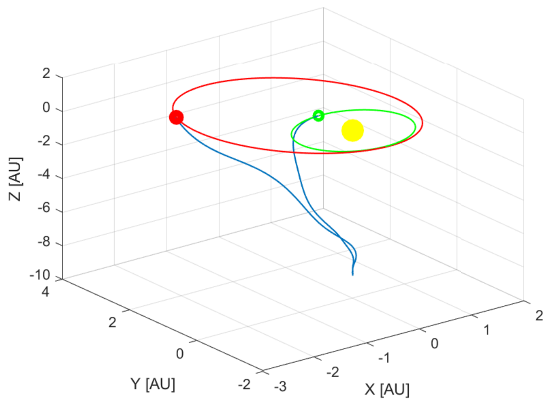
Looking back at the pre-processing process described in Section 9.3.1, it was decided that the following three functions would be used for the different components: a power function (r), a logarithmic function (θ) and finally a sinusoidal function (z). This choice was based on Figure 9.4, which has been generated with the data from the reference case in Table 9.3.



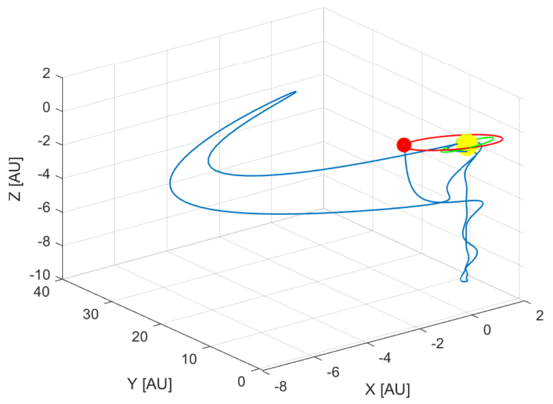
(a) $f(\tau) = a\tau^3 + b\tau^2 + c\tau + d$



(b) $f(\tau) = ae^{b\tau} + c$



(c) $f(\tau) = A\sin\left(\omega\left(\tau - \phi\right)\right) + K$



(d) $f(\tau) = a + b\ln(\tau)$

Figure 11.4: The trajectories to Dionysus computed with four different initialisation functions that were applied to generate an a priori estimate of the r-, θ and z-components.

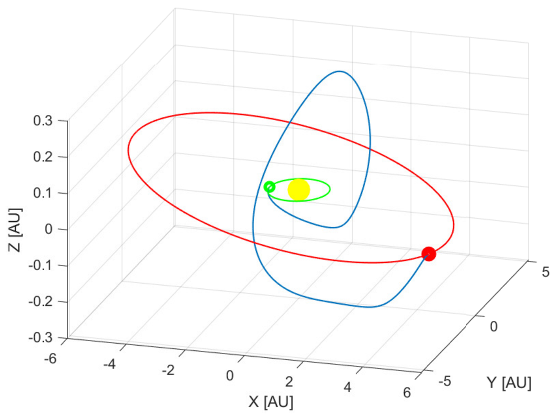
Despite the more accurate tailoring of the initialisation functions to the actual behaviour of the three components, the solver does not converge to a feasible result. Purely based on the ΔV values in Table 11.5, it can be said that the original approach of leaving the approximation in the z-direction untouched does yield a better outcome. However, this is only relative to the other outcome in the row below, because both results are too far off.

Table 11.5: The outcome of the solver when the a priori coefficients are computed by different functions that were applied to the r-, θ and z-components for a trajectory to Jupiter.

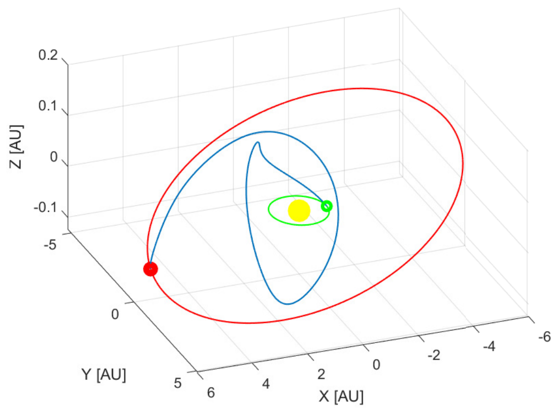
Coordinate	Function	ΔV [km/s]	Best Multi-start ΔV [km/s]
Radius Transfer angle Axial distance	Power Logarithmic -	126.44	457.58
Radius Transfer angle Axial distance	Power Logarithmic Sinusoidal	135.02	385.40

There is no doubt that the computed trajectories in Figure 11.5 are off. Yet, they seem to show other flaws than the odd creations in Figures 11.1 and 11.3. The transfer orbit that is based on two initialisation functions (Figure 11.5a) does have a semi-smooth shape. It shows four kinks, but at these points the trajectory does not change its direction in the completely opposite direction. It is rather deviating its course in an inefficient manner. Also, in terms of movement in the z-direction, it remains within the appropriate range. Nonetheless, it is prone to seriously overshoot and undershoot the target trajectory, again lacking efficiency.

On the other hand, the trajectory that was approximated by three different initialisation functions (Figure 11.5b) shows the same behaviour that was often encountered when too few discretisation points have been used, such as in the two-dimensional version as shown in Section 7.2. The early peak in the trajectory shape can also be observed at the end of the trajectories in Figure 7.7.



(a) The trajectory to Jupiter where r is approximated by a power function, θ by a logarithmic function and where z is set to zero.



(b) The trajectory to Jupiter where r is approximated by a power function, θ by a logarithmic function and where z is approximated by a sinusoidal function.

Figure 11.5: The trajectories to Jupiter computed with different initialisation functions per Fourier series.

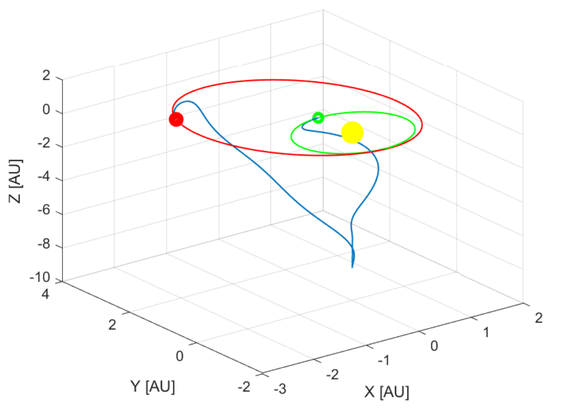
11.3.2. Dionysus

Just as with Jupiter, it was decided that the following three functions would be used for the different components: a power function (r), a logarithmic function (θ) and finally a sinusoidal function (z). This choice was based on Figure 9.6, which has been generated with the data from the reference case in Table 9.6.

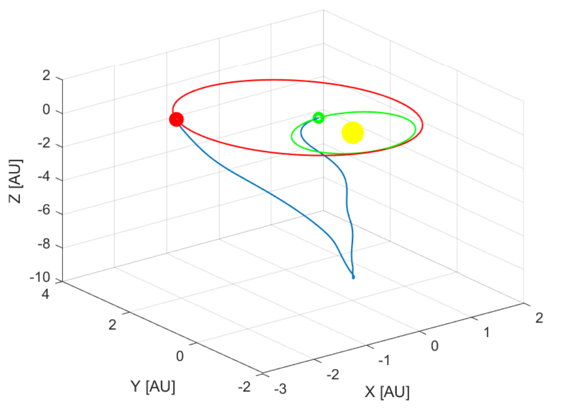
Table 11.6: The outcome of the solver when the a priori coefficients are computed by different functions that were applied to the r-, θ and z-components for a trajectory to Dionysus.

Coordinate	Function	ΔV [km/s]	Best Multi-start ΔV [km/s]
Radius Transfer angle Axial distance	Power Logarithmic -	167.13	2996.02
Radius Transfer angle Axial distance	Power Logarithmic Sinusoidal	170.28	2996.02

Where the finite Fourier series were able to generate a spiralling orbit to reach Jupiter, it is not capable of doing so when computing a trajectory to Dionysus. In Figure 11.6 it can be seen that both trajectories have a similar shape and show identical behaviour: both orbits have a strong tendency to move in the negative z-direction up to approximately 10 AU and they start by thrusting against the direction of motion, which is extremely inefficient in terms of ΔV.



(a) The trajectory to Dionysus where r is approximated by a power function, θ by a logarithmic function and where z is set to zero.



(b) The trajectory to Dionysus where r is approximated by a power function, θ by a logarithmic function and where z is approximated by a sinusoidal function.

Figure 11.6: The trajectories to Dionysus computed with different initialisation functions per Fourier series.

11.4. Coefficient Analysis

Even though a set of Fourier series can be used to capture a low-thrust transfer orbit, the eventual coefficients that determine the shape of this trajectory have no physical meaning. Nonetheless, they do hold valuable information in terms of feasibility and robustness. The coefficient values from the trajectory to Jupiter in Section 11.1 are shown in Figure 11.7 and the ones from the trajectory to Dionysus are shown in Figure 11.8. Note that the first two terms of each cosine or sine series are not included, as these are incorporated by F (recall Equation (4.30a)) and fully determined by the boundary conditions of the trajectory design problem.

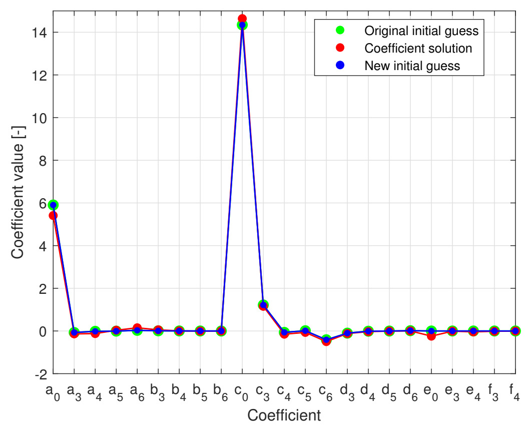
The three colours represent the initial guesses and the final coefficient solution. The green line denotes the reference solution (i.e. the trajectory to Jupiter or Dionysus that has been generated with a power function approximation), while the red line depicts the solution the solver ultimately converged to. The blue line shows the initial guess that has been obtained by means of one of the new initialisation functions.

The first observation that stands out from the four graphs is the dominance of the a_0, c_0 and e_0 terms. For each different initialisation approach, it is a_0, c_0 and e_0 that shows the highest absolute value in the radial, transverse and axial direction, respectively. This is the very first Fourier term of each component and it is the only term that is not affected by the total number of Fourier terms (recall Equation (4.30b)).

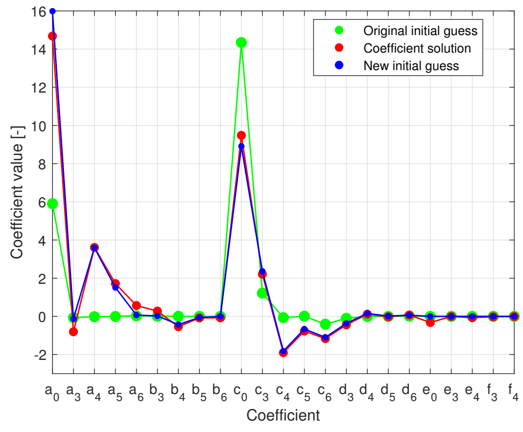
A second look at the figure shows that the reference solution in Figure 11.7a has the smoothest shape of all four. Apart from the aforementioned very first coefficients, the average coefficient value lies around zero. Figures 11.7b to 11.7d show different behaviour. Especially the sinusoidal approximation seems to have a rather random coefficient assignment. It seems as if the coefficients after c_0 need to correct for the large overshoot it demonstrates. The logarithmic function exhibits similar irregular behaviour, but the root-cause of its failure is likely to be the magnitude of the coefficients. The maximum coefficient in Figure 11.7a is only 14.64, whereas the first logarithmic estimate already skyrockets to nearly 700.

The third finding involves the eventual value of the Fourier coefficient that represent the axial position. The two feasible solutions (i.e. Figures 11.7a and 11.7b), show the same e and f coefficients. They are all close to zero and as discussed before, the e_0 has the largest absolute value. This is however not the case for the two infeasible trajectories (i.e. Figures 11.7c and 11.7d). In both these figures, a significant increase in the value of e_0 can be seen. Secondly, the e_4 coefficient has a higher weight as well. It is known that the finite Fourier series is highly sensitive when it comes to the three-dimensional component. Therefore, Figure 11.7 seems to explain why a power and exponential function approximation manage to generate a proper trajectory solution, while a sinusoidal and logarithmic function estimate fail to let the solver converge to a feasible solution: the coefficient values in the z-direction are too high.

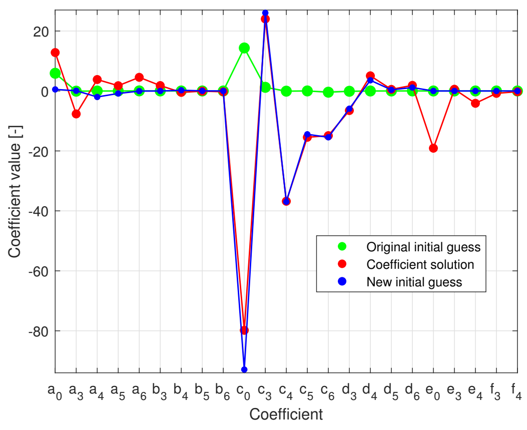
The Fourier coefficients that are used to capture the trajectory to the asteroid Dionysus in Figure 11.8 show an entirely different pattern compared to Figure 11.7. Overall, the same observations regarding the effect of a_0,



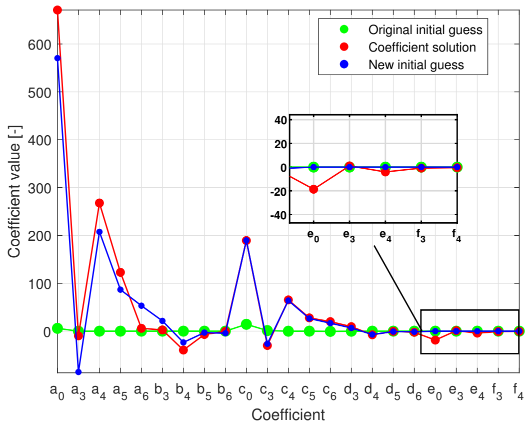
(a) $f(\tau) = a\tau^3 + b\tau^2 + c\tau + d$



(b) $f(\tau) = ae^{b\tau} + c$



(c) $f(\tau) = A\sin\left(\omega\left(\tau - \phi\right)\right) + K$



(d) $f(\tau) = a + b\ln(\tau)$

Figure 11.7: The newly obtained Fourier coefficients and their (original) initial guess for a trajectory to Jupiter. Following the outline of case I, the mentioned function that serves as a subtitle is only used to approximate r and θ. The z coefficients are set equal to zero.

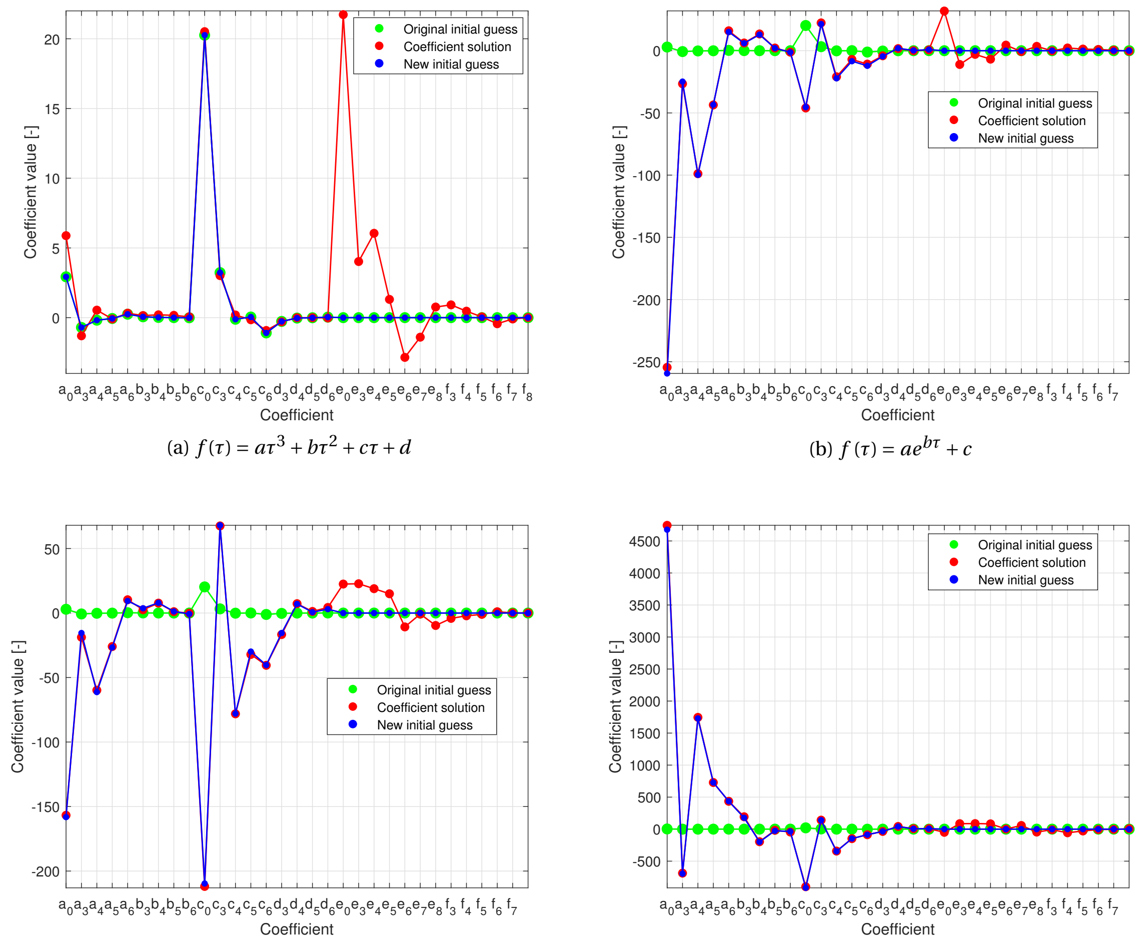
c_0 and e_0, the smoothness of the figure and the sensitivity in the z-direction can be made. However, the scale is much higher and the reference case in Figure 11.8a stands out. The behaviour of the Fourier coefficients of all other three-dimensional simulations can be found in Appendix E

11.5. Multi-start Analysis

The multi-start approach had to be activated 15 times, which is rather often, as only 20 trajectories solutions had to be found. The five trajectory problems that did yield a feasible result the first run only consisted of solutions that were initialised by a power or an exponential function. In Figure 11.9 an overview of the ΔV outcomes of the failed 15 trajectories have been plotted. The title of each sub-figure contains three identifiers: the first one indicates the test object (Jupiter or Dionysus), the second one describes the case type (I, II or III) and the final one contains information about the used initialisation function (power, exponential, sinusoidal or logarithmic). Note that each figure contains the outcome of all 15 multi-start runs.

All graphs in Figure 11.9 show fairly random behaviour and more importantly: none of the multi-starts resulted in a feasible solution. Also, there has not been a single case where the multi-start approach managed to yield a better result than the regular initialisation function, so without the random component.

Even though it might look like the solving process is converging to a minimum, which is especially visible in Figures 11.9d and 11.9h, but also, to a lesser extent, in Figures 11.9a to 11.9c, 11.9e, 11.9g, 11.9k, 11.9n and 11.9o., this is purely coincidental. The solver has no memory concerning previously generated runs, so



(a) $f(\tau) = a\tau^3 + b\tau^2 + c\tau + d$

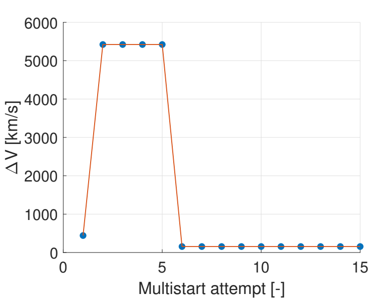
(b) $f(\tau) = ae^{b\tau} + c$

(c) $f(\tau) = A\sin\left(\omega\left(\tau - \phi\right)\right) + K$

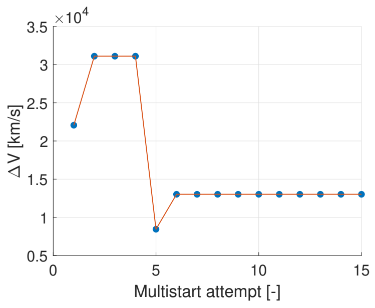
(d) $f(\tau) = a + b\ln(\tau)$

Figure 11.8: The newly obtained Fourier coefficients and their (original) initial guess for a trajectory to Dionysus. Following the outline of case I, the mentioned function that serves as a subtitle is only used to approximate r and θ. The z coefficients are set equal to zero.

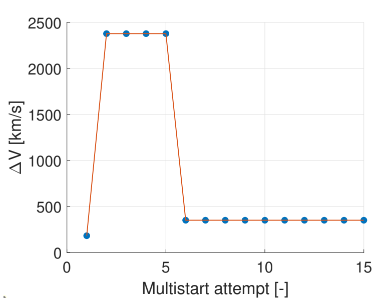
the result from run n is completely independent of the result from run $n-1$. Hence, the newly obtained initial guesses are just within the same region which leads to the same solution.



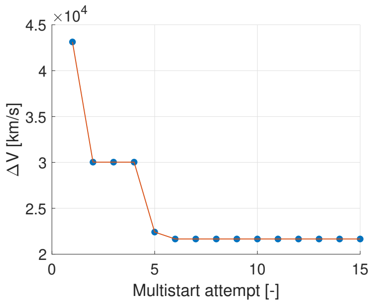
(a) Jupiter - Case I - Sinusoidal



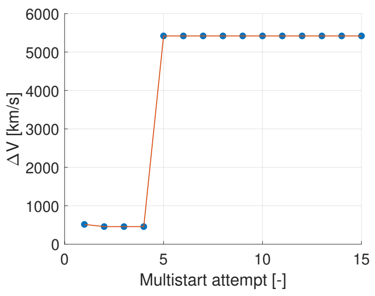
(b) Jupiter - Case I - Logarithmic



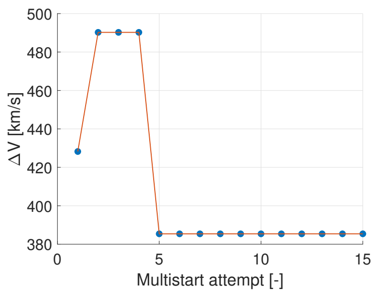
(c) Jupiter - Case II - Sinusoidal



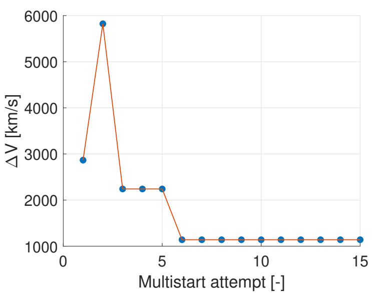
(d) Jupiter - Case II - Logarithmic



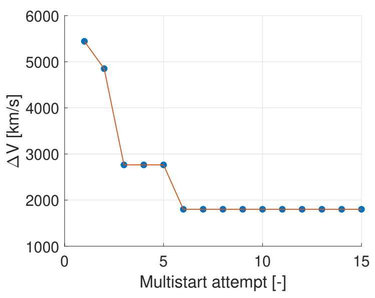
(e) Jupiter - Case III - Polynomial & Logarithmic



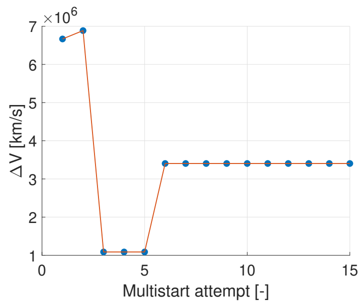
(f) Jupiter - Case III - Polynomial & Logarithmic & Sinusoidal



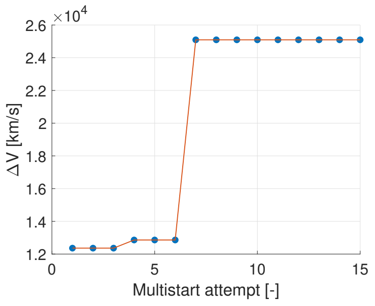
(g) Dionysus - Case I - Exponential



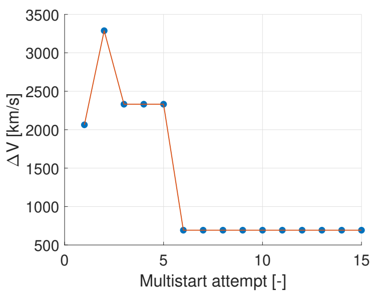
(h) Dionysus - Case I - Sinusoidal



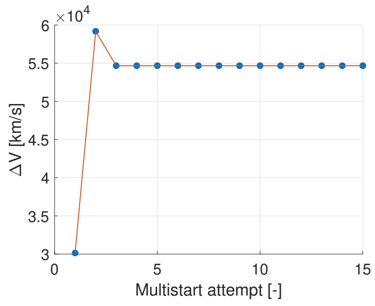
(i) Dionysus - Case I - Logarithmic



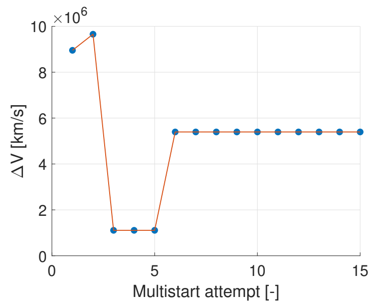
(j) Dionysus - Case II - Power



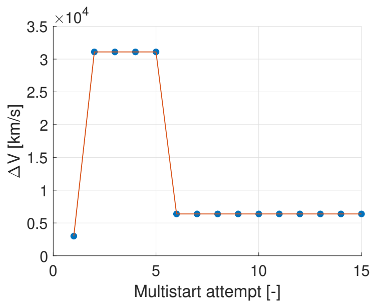
(k) Dionysus - Case II - Exponential



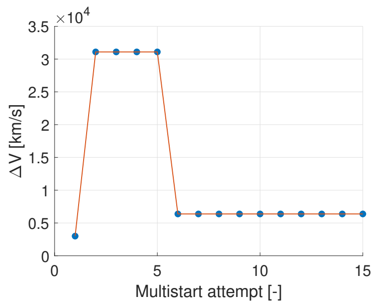
(l) Dionysus - Case II - Sinusoidal



(m) Dionysus - Case II - Logarithmic



(n) Dionysus - Case III - Polynomial & Logarithmic



(o) Dionysus - Case III - Polynomial & Logarithmic & Sinusoidal

Figure 11.9: An overview of the outcome of the 15 multi-start analyses.

12 Conclusions and Recommendations

This final chapter will recapitulate all findings that were uncovered during the implementation, testing and investigation of the finite Fourier series method in Section 12.1. Furthermore, three recommendations are given that might spark interest for future research on this topic in Section 12.2.

12.1. Conclusions

The focus of this thesis was twofold: it covers the implementation of the finite Fourier series method into TUDAT (Technical University of Delft Astrodynamics Toolbox), and it should aim at finding a better initialisation strategy in order to reduce the convergence time of the algorithm. Therefore this conclusion section is divided into three parts. First the discoveries from the implementation phase are presented, followed by the insights obtained from the two-dimensional and three-dimensional attempts of incorporating a new initialisation strategy.

12.1.1. Implementation

The finite Fourier series method by Taheri and Abdelkhalik [21, 22] has proven to be a powerful method when designing low-thrust propulsion orbits. The trajectory is divided into n discretisation points which the equations of motion are evaluated at. By a substitution of a finite Fourier series for r, θ and z, a system of n equations can be said up, which is solved by the Nelder-Mead algorithm to obtain the Fourier coefficients that will eventually shape the trajectory. The equations of motion are kept as simple as possible as it only includes the gravity of the central body and the thrust force, thus neglecting any other perturbation source. Especially its included ability to activate a thrust constraint results in more realistic trajectory solutions compared to other shape-based methods.

The implementation of the method turned out to be more complex than anticipated. This is largely due to ambiguities in the work by Taheri and Abdelkhalik [21, 22]. These ambiguities were found in terms of the generation of the a priori coefficient values, the definition of the decision vector, the performance of the unconstrained finite Fourier series, the two-dimensional results and the interpretation of the reference frame.

For this reason, the matrix system that provides the initial guess had to be redefined. Taheri and Abdelkhalik [21] claim that the system is solved by inverting a non-square matrix. Because this is impossible, the respective matrix has been reformulated such that it is a square one.

Furthermore, the decision vector that is to be altered by the Nelder-Mead solving algorithm was not clearly specified. After testing several implementations, it was found that the approach that does not include the first eight coefficients in the solving process, but rather expresses them as a function of the other coefficients, worked best.

Additionally, it turned out that the unconstrained finite Fourier series fails to converge to a feasible solution. This problem was overcome by adding additional constraints to the transfer angle that specify that the angle at discretisation point n should always be larger than the angle at discretisation point $(n-1)$, enforcing a continuous trajectory.

On top of that, it is not entirely clear with which settings Taheri and Abdelkhalik [21] obtained their results. Nowhere in the paper numerical results are presented. Everything is graphical, which makes it harder to validate the implementation. Also, for a certain number of discretisation points, the solver was not able to converge to a feasible solution. To overcome this problem, a multi-start procedure has been implemented, which entails that 15 different decision vectors are generated in the solution space that might just push the solver into the right direction. These multi-start decision vectors are based on a normal distribution obtained from the first initial guess.

Ultimately, when extracting the position of the departure and arrival bodies from TUDAT, the state vectors are defined in the ecliptic reference frame. When a trajectory is computed, a discrepancy was found between the position of the target body and the endpoint of the transfer orbit at t = TOF. After some more research it turned out that Taheri and Abdelkhalik [21, 22] always normalise their variables such that a transfer angle value of 0° corresponds to a point on the positive x-axis. This has not been mentioned in their work, but it has been solved by adding the angle between the vernal equinox and the departure body at the start of the trajectory.

Two implementation road maps have been set-up to clearly distinguish how the different steps in the computation process are structured in TUDAT, both for the two-dimensional form and the three-dimensional form. The performance of the finite Fourier series method has been validated against previously conducted research that used the method to compute interplanetary transfer orbits and the results appeared to be virtually identical when the tolerances were set to 1×10^{-9}.

It has been discovered that next to the orbital parameters of the departure and target bodies, the outcome of a trajectory design problem is also largely dependent on the number of Fourier series that are used in all three dimensions (i.e. n_r, n_θ and n_z). There is a strong correlation between the number of coefficients and the required ΔV: the more coefficients, the higher the calculated ΔV. The coefficients for the radial component n_r and the transverse component n_θ show the same pattern, but the increase in ΔV is significantly more for the number of coefficients that capture the axial component (n_z). It could therefore be said that the finite Fourier series method is more sensitive in three dimensions, than in two dimensions.

12.1.2. 2D Initialisation

In the search of a better suited initialisation function, four different function types have been analysed: a power function, an exponential function, a trigonometric function and a logarithmic function. When they are applied to estimate both the radial distance and the transfer angle, all four functions manage to generate feasible solutions for which the obtained ΔV are constant at 22.46 km/s for the test case to the asteroid Dionysus, while the outcomes to the planet Jupiter show an average of 18.22 km/s with a spread of only 0.085 km/s.

Of these four initialisation functions it is the exponential function that provides the best a priori results, as it significantly decreases the convergence time of the solver. In order to measure the computation time, the average time of 1000 runs has been taken to minimise the noise that is generated by background processes running on the computer. Whereas the solver requires 312.0 and 278.4 ms with the regular initialisation function to compute a trajectory to Jupiter and Dionysus, it only requires 287.2 and 159.8 ms to do so when an exponential function is used. This is an improvement of 7.9 and 42.6% respectively.
Besides the main solver convergence time, the time to compute all the coefficients of the initialisation functions from the boundary conditions has been assessed in the same way. However, the order of magnitude of this time is only µs. In combination with the very small differences in initialisation time between the four functions, this effect has been deemed negligible.

In case the initialisation function is chosen such that it best fits the natural behaviour of the components, the radial distance r is best approximated by a power function, while the transfer angle θ is best approximated by a logarithmic function. With this set-up, a trajectory to Jupiter was successfully computed, but with 19.0 km/s it requires a higher ΔV than the aforementioned case. It also takes longer (337.6 ms) to find this result. For Dionysus no feasible trajectory could be found.

12.1.3. 3D Initialisation

In three dimensions the same experiments as in two dimensions have been done, but when is referred to the regular solution, the axial coordinates are not estimated by a function, but they are just set equal to zero.

When all three components are modelled by the same initialisation function, it is only the exponential function that finds a feasible trajectory to Jupiter, but instead of the nominal value of 16.7 km/s, it finds a ΔV of 18.6 km/s. None of the functions captures a feasible trajectory to Dionysus, even though a multi-start procedure was used that generates 15 starting points, based on the outcome of the initialisation function.

The tailored-function approach is identical to the two-dimensional case, except for the presence of the axial component, which is now approximated by a sinusoidal function. This strategy too does not yield any feasible result for any of the target bodies, even if the z-coefficients are not approximated and initially set to zero. Just as in the previous case, the multi-start approach did not lead to any feasible results either.

In general, it has been found that the finite Fourier series method is extremely sensitive regarding a priori values for the axial coordinate. If the results from the trajectories to Jupiter and Dionysus from the first case, during which all a priori z-coefficients are set to zero, are compared to the case where they are estimated by an actual initialisation function, much higher ΔV values and even more unrealistic trajectory shapes are obtained.

Main Conclusion

All of the above mentioned findings result in the conclusion that the use of a proper initialisation function can significantly boost the effectiveness of the finite Fourier series method. However, it should be noted that a clear distinction between the two-dimensional and three-dimensional version needs to be made. In two dimensions it can be said that an exponential initialisation function is capable of finding the correct solution in significantly less time. However, in three dimensions no improved initialisation strategy could be found, neither in terms of convergence time nor in stability.

12.2. Recommendations

Based on the generated results for the three-dimensional case, it can be concluded that the currently used optimiser, i.e. Nelder-Mead Simplex Method, is strongly influenced by the a priori information. With four different initialisation functions it could find four different ΔV values. Therefore it is recommended to look into the solving process and the solving technique in particular to make it more robust.

With the current set of initialisation functions, the results are especially effective in two dimensions. In the past, Vroom [25] designed an entirely new function that is used in combination with the spherical shaping method, where it could successfully generate trajectories to targets at an inclination that is above average. However, the computed ΔV did not correspond to these trajectories. Still, it might provide a better initialisation function to be used with the three-dimensional finite Fourier series method.

Just as Taheri and Abdelkhalik [22], in this thesis the recommended number of Fourier terms to model a trajectory is based on trial and error. It has been shown that the number of Fourier coefficients greatly determines the outcome. Hence it is recommended to study this effect to a second degree in order to come up with clear propositions on the number of terms per trajectory design problem.

Bibliography

[1] Detlef Amberg. Fitting a Damped Sine Wave. URL `https://www.dsprelated.com/showarticle/795.php`. Accessed on: 01-05-2020.

[2] Derek Bingham. Schwefel function, February 2020. URL `https://www.sfu.ca/~ssurjano/schwef.html`. Accessed on: 03-02-2020.

[3] J. L. Ortiz et al. The size, shape, density and ring of the dwarf planet Haumea from a stellar occultation. *Nature*, 550(7675):219–223, Oct 2017. doi: 10.1038/nature24051.

[4] B.H. Foing, G.D. Racca, A. Marini, E. Evrard, L. Stagnaro, M. Almeida, D. Koschny, D. Frew, J. Zender, J. Heather, M. Grande, J. Huovelin, H.U. Keller, A. Nathues, J.L. Josset, A. Malkki, W. Schmidt, G. Noci, R. Birkl, L. Iess, Z. Sodnik, and P. McManamon. SMART-1 mission to the Moon: Status, first results and goals. *Advances in Space Research*, 37(1):6–13, January 2006. doi: 10.1016/j.asr.2005.12.016.

[5] D. J. Gondelach and R. Noomen. Hodographic-Shaping Method for Low-Thrust Interplanetary Trajectory Design. *Journal of Spacecraft and Rockets*, 52(3):728–738, May 2015. doi: 10.2514/1.a32991.

[6] David J. Gondelach. A Hodographic-Shaping Method for Low-Thrust Trajectory Design. MSc Thesis, Delft University of Technology, July 2012.

[7] Craig A. Kluever and Steven R. Oleson. Direct Approach for Computing Near-Optimal Low-Thrust Earth-Orbit Transfers. *Journal of Spacecraft and Rockets*, 35(4):509–515, July 1998. doi: 10.2514/2.3360.

[8] Christophe Koppel, Claudio Bruno, Dominique Valentian, Paul Latham, Dave Fearn, and David NICOLINI. Preliminary Comparison Between Nuclear-Electric and Solar-Electric Propulsion Systems for Future Interplanetary Missions. In *39th AIAA/ASME/SAE/ASEE Joint Propulsion Conference and Exhibit*. American Institute of Aeronautics and Astronautics, Jul 2003. doi: 10.2514/6.2003-4689.

[9] NASA's Jet Propulsion Laboratory. Jet Propulsion Laboratory Small-Body Database Browser: 136108 Haumea (2003 EL_{61}), . URL `https://ssd.jpl.nasa.gov/sbdb.cgi?sstr=136108`. Accessed on: 25-08-2018.

[10] NASA's Jet Propulsion Laboratory. Jet Propulsion Laboratory Keplerian Elements for Approximate Positions of the Major Planets, . URL `https://ssd.jpl.nasa.gov/txt/aprx_pos_planets.pdf`. Accessed on: 12-05-2020.

[11] Sjoerd Molenaar. Development of an Improved Spherical Shaping Method for High-Inclination Trajectories. MSc Thesis, Delft University of Technology, August 2009.

[12] D. M. Novak and M. Vasile. Improved Shaping Approach to the Preliminary Design of Low-Thrust Trajectories. *Journal of Guidance, Control, and Dynamics*, 34(1):128–147, January 2011. doi: 10.2514/1.50434.

[13] Masataka Okutsu, Chit Hong Yam, and James Longuski. Low-Thrust Trajectories to Jupiter via Gravity Assists from Venus, Earth, and Mars. In *AIAA/AAS Astrodynamics Specialist Conference and Exhibit*. American Institute of Aeronautics and Astronautics, Aug 2006. doi: 10.2514/6.2006-6745.

[14] Anastassios E. Petropoulos and James M. Longuski. A Shape-Based Algorithm for Automated Design of Low-Thrust, Gravity-Assist Trajectories. *Journal of Spacecraft and Rockets*, 41(5):787–796, September 2004. doi: 10.2514/1.13095.

[15] William H. Press, Saul A. Teukolsky, William T. Vetterling, and Brian P. Flannery. *Numerical Recipes - The Art of Scientific Computing*. Cambridge University Press, New York City, New York, USA, 3^{rd} edition, September 2007. ISBN 9780521880688.

[16] Marc D. Rayman and Steven N. Williams. Design of the First Interplanetary Solar Electric Propulsion Mission. *Journal of Spacecraft and Rockets*, 39(4):589–595, July 2002. doi: 10.2514/2.3848.

[17] Tineke G.R. Roegiers. Application of the Spherical Shaping Method to a Low-Thrust Multiple Asteroid Rendezvous Mission: Implementation, Limitations and Solutions. MSc Thesis, Delft University of Technology, August 2014.

[18] Diogo Sanchez, Antonio F. Prado, Alexander Sukhanov, and Tadashi Yokoyama. Optimal Transfer Trajectories to the Haumea System. In *SpaceOps 2014 Conference*. American Institute of Aeronautics and Astronautics, May 2014. doi: 10.2514/6.2014-1639.

[19] Ernst Stuhlinger. Electric space propulsion systems. *Space Science Reviews*, 7(5-6):795–847, Dec 1967. doi: 10.1007/bf00542896.

[20] Ehsan Taheri. *Rapid Space Trajectory Generation Using a Fourier Series Shape-Based Approach*. PhD Thesis, Michigan Technological University, 2014.

[21] Ehsan Taheri and Ossama Abdelkhalik. Shape-Based Approximation of Constrained Low-Thrust Space Trajectories Using Fourier Series. *Journal of Spacecraft and Rockets*, 49(3):535–545, May 2012. doi: 10.2514/1.a32099.

[22] Ehsan Taheri and Ossama Abdelkhalik. Initial three-dimensional low-thrust trajectory design. *Advances in Space Research*, 57(3):889–903, February 2016. doi: 10.1016/j.asr.2015.11.034.

[23] Ehsan Taheri, Ilya Kolmanovsky, and Ella Atkins. Shaping low-thrust trajectories with thrust-handling feature. *Advances in Space Research*, 61(3):879–890, February 2018. doi: 10.1016/j.asr.2017.11.006.

[24] J. Vlassenbroeck and R. Van Dooren. A Chebyshev technique for solving nonlinear optimal control problems. *IEEE Transactions on Automatic Control*, 33(4):333–340, Apr 1988. doi: 10.1109/9.192187.

[25] Aram Vroom. Development of an Improved Spherical Shaping Method for High-Inclination Trajectories. MSc Thesis, Delft University of Technology, August 2017.

[26] Karel F. Wakker. *Fundamentals of Astrodynamics*. Delft University of Technology, Delft, the Netherlands, 1^{st} edition, January 2015. ISBN 978-94-6186-419-2.

[27] Bradley Wall. Shape-Based Approximation Method for Low-Thrust Trajectory Optimization. In *AIAA/AAS Astrodynamics Specialist Conference and Exhibit*, Honolulu, HA, USA, August 2008. American Institute of Aeronautics and Astronautics. doi: 10.2514/6.2008-6616.

[28] Bradley J. Wall and Bruce A. Conway. Shape-Based Approach to Low-Thrust Rendezvous Trajectory Design. *Journal of Guidance, Control, and Dynamics*, 32(1):95–101, January 2009. doi: 10.2514/1.36848.

[29] Thomas Weise. *Global Optimization Algorithms - Theory and Application* -. self-published, 2^{nd} edition, June 2009.

[30] Barry T.C. Zandbergen. Aerospace Design & Systems Engineering Elements I Part: Spacecraft (bus) design and sizing. Technical report, Delft University of Technology, Faculty of Aerospace Engineering, Delft, the Netherlands, August 2017.

[31] Kui Zeng, Yunhai Geng, Baolin Wu, and Chengqing Xie. A Novel Shape-Based Approximation Method for Constrained Low-Thrust Trajectory Design. In *AIAA/AAS Astrodynamics Specialist Conference*. American Institute of Aeronautics and Astronautics, September 2016. doi: 10.2514/6.2016-5637.

A
Cubic Polynomial Coefficient Derivation

This appendix contains a more extensive derivation of the cubic polynomial coefficients that are used to compute an initial guess for the Fourier coefficients. In Appendix A.1 the derivation of the two-dimensional version is presented, followed by the derivation of the three-dimensional polynomial coefficients in Appendix A.2.

A.1. Real Time

The cubic polynomial that is used for the two-dimensional finite Fourier series implementation is described by the real time t and it is given by the following equations:

$$r_{CP}(t) = at^3 + bt^2 + ct + d \tag{A.1a}$$

$$\theta_{CP}(t) = et^3 + ft^2 + gt + h \tag{A.1b}$$

If these two equations are being derived with respect to time t, the following is obtained:

$$\dot{r}_{CP}(t) = 3at^2 + 2bt + c \tag{A.2a}$$

$$\dot{\theta}_{CP}(t) = 3et^2 + 2ft + g \tag{A.2b}$$

The boundary conditions contain important data for the beginning and the end of the trajectory. The following conditions hold:

$$\begin{cases} r(t=0) &= r_i &= d \\ r(t=t_f) &= r_f &= at_f^3 + bt_f^2 + ct_f + d \\ \dot{r}(t=0) &= \dot{r}_i &= c \\ \dot{r}(t=t_f) &= \dot{r}_f &= 3at_f^2 + 2bt_f + c \end{cases} \tag{A.3a}$$

$$\begin{cases} \theta(t=0) &= \theta_i &= h \\ \theta(t=t_f) &= \theta_f &= et_f^3 + ft_f^2 + gt_f + h \\ \dot{\theta}(t=0) &= \dot{\theta}_i &= g \\ \dot{\theta}(t=t_f) &= \dot{\theta}_f &= 3et_f^2 + 2ft_f + g \end{cases} \tag{A.3b}$$

If the terms in Equation (A.3) are substituted and rearranged, the separate coefficients can be found:

$$a = \frac{2\left(r_i - r_f\right) + \left(\dot{r}_i + \dot{r}_f\right)t_f}{t_f^3} \tag{A.4a}$$

$$b = -\frac{3\left(r_i - r_f\right) + \left(2\dot{r}_i + \dot{r}_f\right) t_f}{t_f^2} \tag{A.4b}$$

$$c = \dot{r}_i \tag{A.4c}$$

$$d = r_i \tag{A.4d}$$

$$e = \frac{2\left(\theta_i - \theta_f\right) + \left(\dot{\theta}_i + \dot{\theta}_f\right) t_f}{t_f^3} \tag{A.4e}$$

$$f = -\frac{3\left(\theta_i - \theta_f\right) + \left(2\dot{\theta}_i + \dot{\theta}_f\right) t_f}{t_f^2} \tag{A.4f}$$

$$g = \dot{\theta}_i \tag{A.4g}$$

$$h = \theta_i \tag{A.4h}$$

A.2. Scaled Time

The cubic polynomial that is used for the three-dimensional finite Fourier series implementation is described by the scaled time τ and it is given by the following equations:

$$r_{CP}(\tau) = a\tau^3 + b\tau^2 + c\tau + d \tag{A.5a}$$

$$\theta_{CP}(\tau) = e\tau^3 + f\tau^2 + g\tau + h \tag{A.5b}$$

If these two equations are being derived with respect to time τ, the following is found:

$$\dot{r}_{CP}(\tau) = 3a\tau^2 + 2b\tau + c \tag{A.6a}$$

$$\dot{\theta}_{CP}(\tau) = 3e\tau^2 + 2f\tau + g \tag{A.6b}$$

Because the time is scaled, the boundary conditions are slightly different from the ones mentioned in Equations (A.3a) and (A.3b). The scaled time boundary conditions are:

$$\left\{\begin{array}{ccccc} r(\tau=0) & = & r_i & = & d \\ r(\tau=1) & = & r_f & = & a\tau^3 + b\tau^2 + c\tau + d \\ r'(\tau=0) & = & r_i' & = & c \\ r'(\tau=1) & = & r_f' & = & 3a\tau^2 + 2b\tau + c \end{array}\right. \tag{A.7a}$$

$$\left\{\begin{array}{ccccc} \theta(\tau=0) & = & \theta_i & = & h \\ \theta(\tau=1) & = & \theta_f & = & e\tau + f\tau^2 + g\tau + h \\ \theta'(\tau=0) & = & \theta_i' & = & g \\ \theta'(\tau=1) & = & \theta_f' & = & 3e\tau^2 + 2f\tau + g \end{array}\right. \tag{A.7b}$$

Rearranging and substituting the expressions in Equation (A.7) results in the following solution for the separate terms:

$$a = r_f' + r_i' + 2\left(r_i - r_f\right) \tag{A.8}$$

$$b = 3\left(r_f - r_i\right) - 2r_i' - r_f' \tag{A.9}$$

$$c = r_i' \tag{A.10}$$

$$d = r_i \tag{A.11}$$

$$e = \theta_f' + \theta_i' + 2\left(\theta_i - \theta_f\right) \tag{A.12}$$

$$f = 3\left(\theta_f - \theta_i\right) - 2\theta_i' - \theta_f' \tag{A.13}$$

$$g = \theta_i' \tag{A.14}$$

$$h = \theta_i \tag{A.15}$$

B

Initial Coefficients Derivation

This appendix contains an overview of the derivation of the first eight Fourier coefficients that can directly be obtained from the imposed boundary conditions. Note that this derivation holds for the two-dimensional version of the finite Fourier series method.

For the sake of clarity, the Fourier series in their regular form are shown once again below.

$$r(t) = \frac{a_0}{2} + \left\{ \sum_{n=1}^{n_r} a_n \cos\left(\frac{n\pi}{T} t\right) + b_n \sin\left(\frac{n\pi}{T} t\right) \right\} \tag{B.1a}$$

$$\theta(t) = \frac{c_0}{2} + \left\{ \sum_{n=1}^{n_\theta} c_n \cos\left(\frac{n\pi}{T} t\right) + d_n \sin\left(\frac{n\pi}{T} t\right) \right\} \tag{B.1b}$$

In order to derive the expressions in Equation (5.13), the derivatives of the Fourier series are needed, as boundary conditions are imposed on them too. The derivatives of Equation (B.1) can be found in Equation (B.2).

$$\dot{r}(t) = \sum_{n=1}^{n_r} \left\{ -a_n \left(\frac{n\pi}{T}\right) \sin\left(\frac{n\pi}{T} t\right) + b_n \left(\frac{n\pi}{T}\right) \cos\left(\frac{n\pi}{T} t\right) \right\} \tag{B.2a}$$

$$\dot{\theta}(t) = \sum_{n=1}^{n_\theta} \left\{ -c_n \left(\frac{n\pi}{T}\right) \sin\left(\frac{n\pi}{T} t\right) + d_n \left(\frac{n\pi}{T}\right) \cos\left(\frac{n\pi}{T} t\right) \right\} \tag{B.2b}$$

With the given boundary conditions as described in Chapter 5, Equations (B.1) and (B.2) can be rewritten as follows by extracting the firs two terms from the summations:

$$r(t_0 = 0) = r_i = \frac{a_0}{2} + \sum_{n=1}^{n_r} = \frac{a_0}{2} + a_1 + a_2 + \sum_{n=3}^{n_r} \{a_n\} \tag{B.3a}$$

$$\theta(t_0 = 0) = \theta_i = \frac{c_0}{2} + \sum_{n=1}^{n_\theta} = \frac{c_0}{2} + c_1 + c_2 + \sum_{n=3}^{n_\theta} \{c_n\} \tag{B.3b}$$

$$\dot{r}(t_0 = 0) = \dot{r}_i = \frac{\pi}{T} + \sum_{n=1}^{n_r} \{n b_n\} = \frac{\pi}{T}(b_1 + 2b_2) + \frac{\pi}{T} \sum_{n=3}^{n_r} \{n b_n\} \tag{B.3c}$$

$$\dot{\theta}(t_0 = 0) = \dot{\theta}_i = \frac{\pi}{T} + \sum_{n=1}^{n_\theta} \{n d_n\} = \frac{\pi}{T}(d_1 + 2d_2) + \frac{\pi}{T} \sum_{n=3}^{n_\theta} \{n d_n\} \tag{B.3d}$$

$$r(t_f = T) = r_f = \frac{a_0}{2} + \sum_{n=1}^{n_r} \left\{(-1)^n a_n\right\} = \frac{a_0}{2} - a_1 + a_2 + \sum_{n=3}^{n_r} \left\{(-1)^n a_n\right\} \tag{B.3e}$$

$$\theta(t_f = T) = \theta_f = \frac{c_0}{2} + \sum_{n=1}^{n_\theta} \left\{(-1)^n c_n\right\} = \frac{c_0}{2} - c_1 + c_2 + \sum_{n=3}^{n_\theta} \left\{(-1)^n c_n\right\} \tag{B.3f}$$

$$\dot{r}\left(t_f = T\right) = \dot{r}_f = \frac{\pi}{T}\sum_{n=1}^{n_r}\left\{(-1)^n n b_n\right\} = \frac{\pi}{T}(2b_2 - b_1) + \frac{\pi}{T}\sum_{n=3}^{n_r}\left\{(-1)^n n b_n\right\} \tag{B.3g}$$

$$\dot{\theta}\left(t_f = T\right) = \dot{\theta}_f = \frac{\pi}{T}\sum_{n=1}^{n_\theta}\left\{(-1)^n n d_n\right\} = \frac{\pi}{T}(2d_2 - d_1) + \frac{\pi}{T}\sum_{n=3}^{n_\theta}\left\{(-1)^n n d_n\right\} \tag{B.3h}$$

From the eight relations in Equation (B.3), four systems of equations can be set up, one for each of the coefficients a_n, b_n, c_n and d_n:

$$\begin{cases} a_1 + a_2 = r_i - \frac{a_0}{2} - \sum_{n=3}^{n_r}\{a_n\} \\ a_2 - a_1 = r_f - \frac{a_0}{2} - \sum_{n=3}^{n_r}\left\{(-1)^n a_n\right\} \end{cases} \tag{B.4a}$$

$$\begin{cases} b_1 + 2b_2 = \frac{T}{\pi}\dot{r}_i - \sum_{n=3}^{n_r}\{n b_n\} \\ 2b_2 - b_1 = \frac{T}{\pi}\dot{r}_f - \sum_{n=3}^{n_r}\left\{(-1)^n b_n\right\} \end{cases} \tag{B.4b}$$

$$\begin{cases} c_1 + c_2 = \theta_i - \frac{c_0}{2} - \sum_{n=3}^{n_\theta}\{c_n\} \\ c_2 - c_1 = \theta_f - \frac{c_0}{2} - \sum_{n=3}^{n_\theta}\left\{(-1)^n c_n\right\} \end{cases} \tag{B.4c}$$

$$\begin{cases} d_1 + 2d_2 = \frac{T}{\pi}\dot{\theta}_i - \sum_{n=3}^{n_\theta}\{n d_n\} \\ 2d_2 - d_1 = \frac{T}{\pi}\dot{\theta}_f - \sum_{n=3}^{n_\theta}\left\{(-1)^n d_n\right\} \end{cases} \tag{B.4d}$$

With the four systems in Equation (B.4), the final solutions for the initial eight coefficients can be computed:

$$a_1 = \frac{r_i - r_f}{2} - \sum_{n=3}^{n_r} a_n;\ n_r \geq 3,\ n: \text{odd} \tag{B.5a}$$

$$a_2 = \frac{r_i + r_f - a_0}{2} - \sum_{n=4}^{n_r} a_n;\ n_r \geq 4,\ n: \text{even} \tag{B.5b}$$

$$b_1 = \frac{T}{2\pi}\left(\dot{r}_i - \dot{r}_f\right) - \sum_{n=3}^{n_r} n b_n;\ n_r \geq 3,\ n: \text{odd} \tag{B.5c}$$

$$b_2 = \frac{T}{4\pi}\left(\dot{r}_i + \dot{r}_f\right) - \frac{1}{2}\sum_{n=4}^{n_r} n b_n;\ n_r \geq 4,\ n: \text{even} \tag{B.5d}$$

$$c_1 = \frac{\theta_i - \theta_f}{2} - \sum_{n=3}^{n_\theta} c_n;\ n_\theta \geq 3,\ n: \text{odd} \tag{B.5e}$$

$$c_2 = \frac{\theta_i + \theta_f - c_0}{2} - \sum_{n=4}^{n_\theta} c_n;\ n_\theta \geq 4,\ n: \text{even} \tag{B.5f}$$

$$d_1 = \frac{T}{2\pi}\left(\dot{\theta}_i - \dot{\theta}_f\right) - \sum_{n=3}^{n_\theta} n d_n;\ n_\theta \geq 3,\ n: \text{odd} \tag{B.5g}$$

$$d_2 = \frac{T}{4\pi}\left(\dot{\theta}_i + \dot{\theta}_f\right) - \frac{1}{2}\sum_{n=4}^{n_\theta} n d_n;\ n_\theta \geq 4,\ n: \text{even} \tag{B.5h}$$

C
3D Finite Fourier Series Derivations

This appendix contains an elaborate overview of the set-up of all terms in the equations for the three-dimensional finite Fourier series method. First, the principle of the scaled time and the conversion between standard time and scaled time is explained in Appendix C.1. In Appendix C.2, the newly adopted notation of the finite Fourier series method is explicated.

C.1. Scaled Time

To understand the conversion from the standard time t, which runs from 0 to T, to the scaled time τ, which runs from 0 to 1, a derivation is given below. Note that τ is defined as follows:

$$\tau = \frac{t}{T} \tag{C.1}$$

The Fourier series for r is shown in Equation (C.2):

$$r(\tau) = \frac{a_0}{2} + \left\{ \sum_{n=1}^{n_r} a_n \cos(n\pi\tau) + b_n \sin(n\pi\tau) \right\} \tag{C.2}$$

Its derivative, r' with respect to τ is then given by Equation (C.3):

$$r'(\tau) = \sum_{n=1}^{n_r} \{-a_n n\pi \sin(n\pi\tau) + b_n n\pi \cos(n\pi\tau)\} \tag{C.3}$$

The derivative of the Fourier series r with respect to t is found in Equation (C.4):

$$\dot{r}(t) = \sum_{n=1}^{n_r} \left\{ -a_n \left(\frac{n\pi}{T}\right) \sin\left(\frac{n\pi}{T} t\right) + b_n \left(\frac{n\pi}{T}\right) \cos\left(\frac{n\pi}{T} t\right) \right\} \tag{C.4}$$

From Equations (C.3) and (C.4), it can be seen that the following relation holds:

$$r'(\tau) = \dot{r}(t) \cdot T \tag{C.5}$$

C.2. Rephrased Fourier Terms

The final equation that needs to be solved to obtain the unknown Fourier coefficients has the following shape:

$$[r]_{m\times1} = [A_r]_{m\times(2n_r-3)} \; [X_r]_{(2n_r-3)\times1} + [F_r]_{m\times1} \tag{C.6}$$

$$[\theta]_{m\times1} = [A_\theta]_{m\times(2n_\theta-3)} \; [X_\theta]_{(2n_\theta-3)\times1} + [F_\theta]_{m\times1} \tag{C.7}$$

$$[z]_{m\times1} = [A_z]_{m\times(2n_z-3)} \; [X_z]_{(2n_z-3)\times1} + [F_z]_{m\times1} \tag{C.8}$$

In which the $[A]$ matrices contain the sine and cosine components of the Fourier series. The boundary conditions are collected in the $[F]$ vectors, the actual coordinates can be found in the $[r]$, $[\theta]$ and $[z]$ vectors, while the Fourier terms are defined in the $[X_r]$, $[X_\theta]$ and $[X_z]$ vectors.

So, instead of computing the first two coefficients of each Fourier terms as in the two-dimensional case, the initial coefficients are all collected in one term: F_r, F_θ or F_z. They are a function of the boundary conditions defined in each of the three directions. Equation (C.9) denotes the expressions that contain all the a priori known information, together with their first and second derivatives with respect to the scaled time τ:

$$F_r = \frac{1}{2}\left(r_i - r_f\right)\cos(\pi\tau) + \frac{1}{2\pi}\left(r'_i - r'_f\right)\sin(\pi\tau) + \frac{1}{2}\left(r_i + r_f\right)\cos(2\pi\tau) + \frac{1}{4\pi}\left(r'_i + r'_f\right)\sin(2\pi\tau) \tag{C.9a}$$

$$F'_r = -\frac{1}{2}\left(r_i - r_f\right)\pi\sin(\pi\tau) + \frac{1}{2}\left(r'_i - r'_f\right)\cos(\pi\tau) - \left(r_i + r_f\right)\pi\sin(2\pi\tau) + \frac{1}{2}\left(r'_i + r'_f\right)\cos(2\pi\tau) \tag{C.9b}$$

$$F''_r = -\frac{\pi^2}{2}\left(r_i - r_f\right)\cos(\pi\tau) - \frac{\pi}{2}\left(r'_i - r'_f\right)\sin(\pi\tau) - 2\pi^2\left(r_i + r_f\right)\cos(2\pi\tau) - \pi\left(r'_i + r'_f\right)\sin(2\pi\tau) \tag{C.9c}$$

$$F_\theta = \frac{1}{2}\left(\theta_i - \theta_f\right)\cos(\pi\tau) + \frac{1}{2\pi}\left(\theta'_i - \theta'_f\right)\sin(\pi\tau) + \frac{1}{2}\left(\theta_i + \theta_f\right)\cos(2\pi\tau) + \frac{1}{4\pi}\left(\theta'_i + \theta'_f\right)\sin(2\pi\tau) \tag{C.9d}$$

$$F'_\theta = -\frac{\pi}{2}\left(\theta_i - \theta_f\right)\sin(\pi\tau) + \frac{1}{2}\left(\theta'_i - \theta'_f\right)\cos(\pi\tau) - \pi\left(\theta_i + \theta_f\right)\sin(2\pi\tau) + \frac{1}{2}\left(\theta'_i + \theta'_f\right)\cos(2\pi\tau) \tag{C.9e}$$

$$F''_\theta = -\frac{\pi^2}{2}\left(\theta_i - \theta_f\right)\cos(\pi\tau) - \frac{\pi}{2}\left(\theta'_i - \theta'_f\right)\sin(\pi\tau) - 2\pi^2\left(\theta_i + \theta_f\right)\cos(2\pi\tau) - \pi\left(\theta'_i + \theta'_f\right)\sin(2\pi\tau) \tag{C.9f}$$

$$F_z = \frac{1}{2}\left(z_i - z_f\right)\cos(\pi\tau) + \frac{1}{2\pi}\left(z'_i - z'_f\right)\sin(\pi\tau) + \frac{1}{2}\left(z_i + z_f\right)\cos(2\pi\tau) + \frac{1}{4\pi}\left(z'_i + z'_f\right)\sin(2\pi\tau) \tag{C.9g}$$

$$F'_z = -\frac{1}{2}\left(z_i - z_f\right)\pi\sin(\pi\tau) + \frac{1}{2}\left(z'_i - z'_f\right)\cos(\pi\tau) - \left(z_i + z_f\right)\pi\sin(2\pi\tau) + \frac{1}{2}\left(z'_i + z'_f\right)\cos(2\pi\tau) \tag{C.9h}$$

$$F''_z = -\frac{\pi^2}{2}\left(z_i - z_f\right)\cos(\pi\tau) - \frac{\pi}{2}\left(z'_i - z'_f\right)\sin(\pi\tau) - 2\pi^2\left(z_i + z_f\right)\cos(2\pi\tau) - \pi\left(z'_i + z'_f\right)\sin(2\pi\tau) \tag{C.9i}$$

Eventually, the trajectory is divided into m segments, depending on the number of discretisation points. Hence, in order to find the Fourier coefficients, Equation (C.9) needs to be evaluated at m points. These values are then stored in a column vector, as can be seen in Equation (C.10):

$$F_r = \begin{bmatrix} F_r(\tau_0) & F_r(\tau_1) & \cdots & F_r(\tau_m) \end{bmatrix}^T \tag{C.10a}$$

$$F'_r = \begin{bmatrix} F'_r(\tau_0) & F'_r(\tau_1) & \cdots & F'_r(\tau_m) \end{bmatrix}^T \tag{C.10b}$$

$$F''_r = \begin{bmatrix} F''_r(\tau_0) & F''_r(\tau_1) & \cdots & F''_r(\tau_m) \end{bmatrix}^T \tag{C.10c}$$

$$F_\theta = \begin{bmatrix} F_\theta(\tau_0) & F_\theta(\tau_1) & \cdots & F_\theta(\tau_m) \end{bmatrix}^T \tag{C.10d}$$

$$F'_\theta = \begin{bmatrix} F'_\theta(\tau_0) & F'_\theta(\tau_1) & \cdots & F'_\theta(\tau_m) \end{bmatrix}^T \tag{C.10e}$$

$$F''_\theta = \begin{bmatrix} F''_r(\tau_0) & F''_r(\tau_1) & \cdots & F''_r(\tau_m) \end{bmatrix}^T \tag{C.10f}$$

$$F_z = \begin{bmatrix} F_r(\tau_0) & F_r(\tau_1) & \cdots & F_r(\tau_m) \end{bmatrix}^T \tag{C.10g}$$

$$F'_z = \begin{bmatrix} F'_r(\tau_0) & F'_r(\tau_1) & \cdots & F'_r(\tau_m) \end{bmatrix}^T \tag{C.10h}$$

$$F''_z = \begin{bmatrix} F''_r(\tau_0) & F''_r(\tau_1) & \cdots & F''_r(\tau_m) \end{bmatrix}^T \tag{C.10i}$$

Now all $[F]$ vectors are set up, the $[A]$ matrices can be filled. The number of columns depends on the number of Fourier series that are used to describe the transfer orbit according to $(2n_r - 3)$, as the first two coefficients of each sine and cosine term are already computed and stored in the $[F]$ vector. The number of rows depends on the number of discretisation points m that have been used. The $[A]$ matrices and their derivatives for r are shown in Equation (C.11):

$$A_r = \begin{bmatrix} C_{a_0}(\tau_0) & C_{a_3}(\tau_0) & C_{b_3}(\tau_0) & C_{a_4}(\tau_0) & C_{b_4}(\tau_0) & \cdots & C_{a_{n_r}}(\tau_0) & C_{b_{n_r}}(\tau_0) \\ C_{a_0}(\tau_1) & C_{a_3}(\tau_1) & C_{b_3}(\tau_1) & C_{a_4}(\tau_1) & C_{b_4}(\tau_1) & \cdots & C_{a_{n_r}}(\tau_1) & C_{b_{n_r}}(\tau_1) \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \cdots & \vdots \\ C_{a_0}(\tau_m) & C_{a_3}(\tau_m) & C_{b_3}(\tau_m) & C_{a_4}(\tau_m) & C_{b_4}(\tau_m) & \cdots & C_{a_{n_r}}(\tau_m) & C_{b_{n_r}}(\tau_m) \end{bmatrix} \tag{C.11a}$$

$$A'_r = \begin{bmatrix} C'_{a_0}(\tau_0) & C'_{a_3}(\tau_0) & C'_{b_3}(\tau_0) & C'_{a_4}(\tau_0) & C'_{b_4}(\tau_0) & \cdots & C'_{a_{n_r}}(\tau_0) & C'_{b_{n_r}}(\tau_0) \\ C'_{a_0}(\tau_1) & C'_{a_3}(\tau_1) & C'_{b_3}(\tau_1) & C'_{a_4}(\tau_1) & C'_{b_4}(\tau_1) & \cdots & C'_{a_{n_r}}(\tau_1) & C'_{b_{n_r}}(\tau_1) \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \cdots & \vdots \\ C'_{a_0}(\tau_m) & C'_{a_3}(\tau_m) & C'_{b_3}(\tau_m) & C'_{a_4}(\tau_m) & C'_{b_4}(\tau_m) & \cdots & C'_{a_{n_r}}(\tau_m) & C'_{b_{n_r}}(\tau_m) \end{bmatrix} \tag{C.11b}$$

$$A''_r = \begin{bmatrix} C''_{a_0}(\tau_0) & C''_{a_3}(\tau_0) & C''_{b_3}(\tau_0) & C''_{a_4}(\tau_0) & C''_{b_4}(\tau_0) & \cdots & C''_{a_{n_r}}(\tau_0) & C''_{b_{n_r}}(\tau_0) \\ C''_{a_0}(\tau_1) & C''_{a_3}(\tau_1) & C''_{b_3}(\tau_1) & C''_{a_4}(\tau_1) & C''_{b_4}(\tau_1) & \cdots & C''_{a_{n_r}}(\tau_1) & C''_{b_{n_r}}(\tau_1) \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \cdots & \vdots \\ C''_{a_0}(\tau_m) & C''_{a_3}(\tau_m) & C''_{b_3}(\tau_m) & C''_{a_4}(\tau_m) & C''_{b_4}(\tau_m) & \cdots & C''_{a_{n_r}}(\tau_m) & C''_{b_{n_r}}(\tau_m) \end{bmatrix} \tag{C.11c}$$

The terms in Equation (C.11) and their derivatives are found in Equation (C.12):

$$C_{a_0} = \frac{1}{2}[1 - \cos(2\pi\tau)] \tag{C.12a}$$

$$C_{a_n} = \begin{cases} \cos(n\pi\tau) - \cos(\pi\tau); & \text{when } n \text{ is odd} \\ \cos(n\pi\tau) - \cos(2\pi\tau); & \text{when } n \text{ is even} \end{cases} \tag{C.12b}$$

$$C_{b_n} = \begin{cases} \sin(n\pi\tau) - n\sin(\pi\tau); & \text{when } n \text{ is odd} \\ \sin(n\pi\tau) - \frac{n}{2}\sin(2\pi\tau); & \text{when } n \text{ is even} \end{cases} \tag{C.12c}$$

$$C'_{a_0} = \pi\sin(2\pi\tau) \tag{C.12d}$$

$$C'_{a_n} = \begin{cases} -n\pi\sin(n\pi\tau) + \pi\sin(\pi\tau); & \text{when } n \text{ is odd} \\ -n\pi\sin(n\pi\tau) + 2\pi\sin(2\pi\tau); & \text{when } n \text{ is even} \end{cases} \tag{C.12e}$$

$$C'_{b_n} = \begin{cases} n\pi\cos(n\pi\tau) - n\pi\cos(\pi\tau); & \text{when } n \text{ is odd} \\ n\pi\cos(n\pi\tau) - n\pi\cos(2\pi\tau); & \text{when } n \text{ is even} \end{cases} \tag{C.12f}$$

$$C''_{a_0} = 2\pi^2\cos(2\pi\tau) \tag{C.12g}$$

$$C''_{a_n} = \begin{cases} -(n\pi)^2\cos(n\pi\tau) + \pi^2\cos(\pi\tau); & \text{when } n \text{ is odd} \\ -(n\pi)^2\cos(n\pi\tau) + 4\pi^2\cos(2\pi\tau); & \text{when } n \text{ is even} \end{cases} \tag{C.12h}$$

$$C''_{b_n} = \begin{cases} -(n\pi)^2\sin(n\pi\tau) + n\pi^2\sin(\pi\tau); & \text{when } n \text{ is odd} \\ -(n\pi)^2\sin(n\pi\tau) + 2n\pi^2\sin(2\pi\tau); & \text{when } n \text{ is even} \end{cases} \tag{C.12i}$$

To fill the $[A]$ matrices for θ, the same procedure applies as for r:

$$A_\theta = \begin{bmatrix} C_{c0}(\tau_0) & C_{c3}(\tau_0) & C_{d3}(\tau_0) & C_{c4}(\tau_0) & C_{d4}(\tau_0) & \cdots & C_{cn_\theta}(\tau_0) & C_{dn_\theta}(\tau_0) \\ C_{c0}(\tau_1) & C_{c3}(\tau_1) & C_{d3}(\tau_1) & C_{c4}(\tau_1) & C_{d4}(\tau_1) & \cdots & C_{cn_\theta}(\tau_1) & C_{dn_\theta}(\tau_1) \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \cdots & \vdots \\ C_{c0}(\tau_m) & C_{c3}(\tau_m) & C_{d3}(\tau_m) & C_{c4}(\tau_m) & C_{d4}(\tau_m) & \cdots & C_{cn_\theta}(\tau_m) & C_{dn_\theta}(\tau_m) \end{bmatrix} \tag{C.13a}$$

$$A'_\theta = \begin{bmatrix} C'_{c0}(\tau_0) & C'_{c3}(\tau_0) & C'_{d3}(\tau_0) & C'_{c4}(\tau_0) & C'_{d4}(\tau_0) & \cdots & C'_{cn_r}(\tau_0) & C'_{dn_r}(\tau_0) \\ C'_{c0}(\tau_1) & C'_{c3}(\tau_1) & C'_{d3}(\tau_1) & C'_{c4}(\tau_1) & C'_{d4}(\tau_1) & \cdots & C'_{cn_r}(\tau_1) & C'_{dn_r}(\tau_1) \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \cdots & \vdots \\ C'_{c0}(\tau_m) & C'_{c3}(\tau_m) & C'_{d3}(\tau_m) & C'_{c4}(\tau_m) & C'_{d4}(\tau_m) & \cdots & C'_{cn_r}(\tau_m) & C'_{dn_r}(\tau_m) \end{bmatrix} \tag{C.13b}$$

$$A''_\theta = \begin{bmatrix} C''_{c0}(\tau_0) & C''_{c3}(\tau_0) & C''_{d3}(\tau_0) & C''_{c4}(\tau_0) & C''_{d4}(\tau_0) & \cdots & C''_{cn_r}(\tau_0) & C''_{dn_r}(\tau_0) \\ C''_{c0}(\tau_1) & C''_{c3}(\tau_1) & C''_{d3}(\tau_1) & C''_{c4}(\tau_1) & C''_{d4}(\tau_1) & \cdots & C''_{cn_r}(\tau_1) & C''_{dn_r}(\tau_1) \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \cdots & \vdots \\ C''_{c0}(\tau_m) & C''_{c3}(\tau_m) & C''_{d3}(\tau_m) & C''_{c4}(\tau_m) & C''_{d4}(\tau_m) & \cdots & C''_{cn_r}(\tau_m) & C''_{dn_r}(\tau_m) \end{bmatrix} \tag{C.13c}$$

which are built up from the following equations:

$$C_{c_0} = \frac{1}{2}[1 - \cos(2\pi\tau)] \tag{C.14a}$$

$$C_{c_n} = \begin{cases} \cos(n\pi\tau) - \cos(\pi\tau); & \text{when } n \text{ is odd} \\ \cos(n\pi\tau) - \cos(2\pi\tau); & \text{when } n \text{ is even} \end{cases} \tag{C.14b}$$

$$C_{d_n} = \begin{cases} \sin(n\pi\tau) - n\sin(\pi\tau); & \text{when } n \text{ is odd} \\ \sin(n\pi\tau) - \frac{n}{2}\sin(2\pi\tau); & \text{when } n \text{ is even} \end{cases} \tag{C.14c}$$

$$C'_{c_0} = \pi\sin(2\pi\tau) \tag{C.14d}$$

$$C'_{c_n} = \begin{cases} -n\pi\sin(n\pi\tau) + \pi\sin(\pi\tau); & \text{when } n \text{ is odd} \\ -n\pi\sin(n\pi\tau) + 2\pi\sin(2\pi\tau); & \text{when } n \text{ is even} \end{cases} \tag{C.14e}$$

$$C'_{d_n} = \begin{cases} n\pi\cos(n\pi\tau) - n\pi\cos(\pi\tau); & \text{when } n \text{ is odd} \\ n\pi\cos(n\pi\tau) - n\pi\cos(2\pi\tau); & \text{when } n \text{ is even} \end{cases} \tag{C.14f}$$

$$C''_{c_0} = 2\pi^2\cos(2\pi\tau) \tag{C.14g}$$

$$C''_{c_n} = \begin{cases} -(n\pi)^2\cos(n\pi\tau) + \pi^2\cos(\pi\tau); & \text{when } n \text{ is odd} \\ -(n\pi)^2\cos(n\pi\tau) + 4\pi^2\cos(2\pi\tau); & \text{when } n \text{ is even} \end{cases} \tag{C.14h}$$

$$C''_{d_n} = \begin{cases} -(n\pi)^2\sin(n\pi\tau) + n\pi^2\sin(\pi\tau); & \text{when } n \text{ is odd} \\ -(n\pi)^2\sin(n\pi\tau) + 2n\pi^2\sin(2\pi\tau); & \text{when } n \text{ is even} \end{cases} \tag{C.14i}$$

To fill the $[A]$ matrices for z, again, the same procedure applies as for r:

$$A_z = \begin{bmatrix} C_{e_0}(\tau_0) & C_{e_3}(\tau_0) & C_{f_3}(\tau_0) & C_{e_4}(\tau_0) & C_{f_4}(\tau_0) & \cdots & C_{e_{n_r}}(\tau_0) & C_{f_{n_r}}(\tau_0) \\ C_{e_0}(\tau_1) & C_{e_3}(\tau_1) & C_{f_3}(\tau_1) & C_{e_4}(\tau_1) & C_{f_4}(\tau_1) & \cdots & C_{e_{n_r}}(\tau_1) & C_{f_{n_r}}(\tau_1) \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \cdots & \vdots \\ C_{e_0}(\tau_m) & C_{e_3}(\tau_m) & C_{f_3}(\tau_m) & C_{e_4}(\tau_m) & C_{f_4}(\tau_m) & \cdots & C_{e_{n_r}}(\tau_m) & C_{f_{n_r}}(\tau_m) \end{bmatrix} \tag{C.15a}$$

$$A'_z = \begin{bmatrix} C'_{e_0}(\tau_0) & C'_{e_3}(\tau_0) & C'_{f_3}(\tau_0) & C'_{e_4}(\tau_0) & C'_{f_4}(\tau_0) & \cdots & C'_{e_{n_z}}(\tau_0) & C'_{f_{n_z}}(\tau_0) \\ C'_{e_0}(\tau_1) & C'_{e_3}(\tau_1) & C'_{f_3}(\tau_1) & C'_{e_4}(\tau_1) & C'_{f_4}(\tau_1) & \cdots & C'_{e_{n_z}}(\tau_1) & C'_{f_{n_z}}(\tau_1) \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \cdots & \vdots \\ C'_{e_0}(\tau_m) & C'_{e_3}(\tau_m) & C'_{f_3}(\tau_m) & C'_{e_4}(\tau_m) & C'_{f_4}(\tau_m) & \cdots & C'_{e_{n_z}}(\tau_m) & C'_{f_{n_z}}(\tau_m) \end{bmatrix} \tag{C.15b}$$

$$A''_z = \begin{bmatrix} C''_{e_0}(\tau_0) & C''_{e_3}(\tau_0) & C''_{f_3}(\tau_0) & C''_{e_4}(\tau_0) & C''_{f_4}(\tau_0) & \cdots & C''_{e_{n_r}}(\tau_0) & C''_{f_{n_z}}(\tau_0) \\ C''_{e_0}(\tau_1) & C''_{e_3}(\tau_1) & C''_{f_3}(\tau_1) & C''_{e_4}(\tau_1) & C''_{f_4}(\tau_1) & \cdots & C''_{e_{n_z}}(\tau_1) & C''_{f_{n_z}}(\tau_1) \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \cdots & \vdots \\ C''_{e_0}(\tau_m) & C''_{e_3}(\tau_m) & C''_{f_3}(\tau_m) & C''_{e_4}(\tau_m) & C''_{f_4}(\tau_m) & \cdots & C''_{e_{n_z}}(\tau_m) & C''_{f_{n_z}}(\tau_m) \end{bmatrix} \tag{C.15c}$$

in which the C terms are defined according to:

$$C_{e_0} = \frac{1}{2}[1 - \cos(2\pi\tau)] \tag{C.16a}$$

$$C_{e_n} = \begin{cases} \cos(n\pi\tau) - \cos(\pi\tau); & \text{when } n \text{ is odd} \\ \cos(n\pi\tau) - \cos(2\pi\tau); & \text{when } n \text{ is even} \end{cases} \tag{C.16b}$$

$$C_{f_n} = \begin{cases} \sin(n\pi\tau) - n\sin(\pi\tau); & \text{when } n \text{ is odd} \\ \sin(n\pi\tau) - \frac{n}{2}\sin(2\pi\tau); & \text{when } n \text{ is even} \end{cases} \tag{C.16c}$$

$$C'_{e_0} = \pi\sin(2\pi\tau) \tag{C.16d}$$

$$C'_{e_n} = \begin{cases} -n\pi\sin(n\pi\tau) + \pi\sin(\pi\tau); & \text{when } n \text{ is odd} \\ -n\pi\sin(n\pi\tau) + 2\pi\sin(2\pi\tau); & \text{when } n \text{ is even} \end{cases} \tag{C.16e}$$

$$C'_{f_n} = \begin{cases} n\pi\cos(n\pi\tau) - n\pi\cos(\pi\tau); & \text{when } n \text{ is odd} \\ n\pi\cos(n\pi\tau) - n\pi\cos(2\pi\tau); & \text{when } n \text{ is even} \end{cases} \tag{C.16f}$$

$$C''_{e_0} = 2\pi^2\cos(2\pi\tau) \tag{C.16g}$$

$$C''_{e_n} = \begin{cases} -(n\pi)^2\cos(n\pi\tau) + \pi^2\cos(\pi\tau); & \text{when } n \text{ is odd} \\ -(n\pi)^2\cos(n\pi\tau) + 4\pi^2\cos(2\pi\tau); & \text{when } n \text{ is even} \end{cases} \tag{C.16h}$$

$$C''_{f_n} = \begin{cases} -(n\pi)^2\sin(n\pi\tau) + n\pi^2\sin(\pi\tau); & \text{when } n \text{ is odd} \\ -(n\pi)^2\sin(n\pi\tau) + 2n\pi^2\sin(2\pi\tau); & \text{when } n \text{ is even} \end{cases} \tag{C.16i}$$

D Implementation Issues

This appendix serves as a succinct overview of all encountered hurdles as an aid for possible future students and because it has led to a significant delay during the implementation of the methods. It will contain an enumeration of all discrepancies, unknowns and vague definitions that have been encountered during the implementation of the two-dimensional and the three-dimensional finite Fourier series described in the publications by Taheri and Abdelkhalik [21, 22]. Issues have been encountered regarding the computation of the initial guess, the definition of the decision vector, discrepancies involving the unconstrained finite Fourier series, in the validation, the solving procedure and the reference frame, which can be found in Appendices D.1 to D.6, respectively. It should be noted that everything that is mentioned in this chapter has already been stated in the main body of this document, mainly in Section 4.3 and chapters 7 and 8.

D.1. Initial Guess Calculation

The initial guess for the solving algorithm for both r and θ is found by a cubic polynomial approximation:

$$r_{CP}(t) = at^3 + bt^2 + ct + d \tag{D.1a}$$

$$\theta_{CP}(t) = et^3 + ft^2 + gt + h \tag{D.1b}$$

In this equation, the eight boundary conditions will be used to compute the value of the eight polynomial coefficients. With the Fourier expressions, as seen in Equation (D.2), Taheri and Abdelkhalik claim that a linear system of the form $A\mathbf{x} = B$ can be set up, in which the Fourier coefficient vector $\mathbf{x}$ (Equation (D.4)) is found by left-multiplying both sides with A^{-1}.

$$r(t) = \frac{a_0}{2} + \left\{ \sum_{n=1}^{n_r} a_n \cos\left(\frac{n\pi}{T}t\right) + b_n \sin\left(\frac{n\pi}{T}t\right) \right\} \tag{D.2a}$$

$$\theta(t) = \frac{c_0}{2} + \left\{ \sum_{n=1}^{n_\theta} c_n \cos\left(\frac{n\pi}{T}t\right) + d_n \sin\left(\frac{n\pi}{T}t\right) \right\} \tag{D.2b}$$

In this expression, the trigonometric terms from Equation (D.2a) are collected in A, which leads to Equation (D.3). The size of this matrix then partly depends on the discretisation of the time and the number of Fourier terms n_r. The number of columns is set by n_r and is equal to (2 x n_r + 1) in order to include the constant $\frac{1}{2}$ term together with n_r cosine terms representing the a_n coefficients, and n_r sine terms representing the b_n coefficients. On the other hand, the number of rows depends on the time discretisation and it is at this point where Taheri and Abdelkhalik went wrong.

Taheri and Abdelkhalik [21] state that the time should be discretised by n_r points, which indicates that matrix A becomes a n_r x ($2n_r$ + 1) matrix. This is not a square matrix, so it is impossible to solve the $A\mathbf{x} = B$ by just left-multiplying both sides with A^{-1}. To overcome this complication, two methods can be applied: either the system needs to be solved by means of finding the least-squares solution, or the dimensions of matrix A should be altered such that it actually becomes a square matrix. It has been decided to go with the latter option. Instead of discretising the total flight time by n_r, it is simply divided into $(2n_r+1)$ points. Equations (D.3) to (D.5) below show the correct definitions and sizes of the matrices. Note that in this case t_i represents the $(2n_r+1)^{th}$ timestep.

$$A = \begin{bmatrix} \frac{1}{2} & \cos\left(\frac{\pi}{T}t_0\right) & \cos\left(\frac{2\pi}{T}t_0\right) & \cdots & \cos\left(\frac{n_r\pi}{T}t_0\right) & \sin\left(\frac{\pi}{T}t_0\right) & \sin\left(\frac{2\pi}{T}t_0\right) & \cdots & \sin\left(\frac{n_r\pi}{T}t_0\right) \\ \frac{1}{2} & \cos\left(\frac{\pi}{T}t_1\right) & \cos\left(\frac{2\pi}{T}t_1\right) & \cdots & \cos\left(\frac{n_r\pi}{T}t_1\right) & \sin\left(\frac{\pi}{T}t_1\right) & \sin\left(\frac{2\pi}{T}t_1\right) & \cdots & \sin\left(\frac{n_r\pi}{T}t_1\right) \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ \frac{1}{2} & \cos\left(\frac{\pi}{T}t_i\right) & \cos\left(\frac{2\pi}{T}t_i\right) & \cdots & \cos\left(\frac{n_r\pi}{T}t_i\right) & \sin\left(\frac{\pi}{T}t_i\right) & \sin\left(\frac{2\pi}{T}t_i\right) & \cdots & \sin\left(\frac{n_r\pi}{T}t_i\right) \end{bmatrix} \tag{D.3}$$

$$\mathbf{x} = \begin{bmatrix} a_0 & a_1 & a_2 & \cdots & a_{n_r} & \cdots & b_1 & b_2 & \cdots & b_{n_r} \end{bmatrix}^T \tag{D.4}$$

$$B = \begin{bmatrix} r_{CP}(t_0) & r_{CP}(t_1) & \cdots & r_{CP}(t_i) \end{bmatrix}^T \tag{D.5}$$

D.2. Decision Vector Definition

Before the right implementation method as described in Chapter 5 was found, another approach had been taken, as Taheri and Abdelkhalik are not entirely clear about their approach. In their paper it is written that the first two coefficients of each sine and cosine term in the Fourier series (i.e. a_1, a_2, b_1, b_2, c_1, c_2, d_1 and d_2) can be written as a function of the boundary conditions and the other Fourier coefficients (in case n_r or n_θ) is larger than two [21]. Especially the latter statement made it seem more straightforward to simply include all Fourier terms in the decision vector. Following this method, the decision vector was defined as in Equation (D.6) below:

$$\mathbf{x} = \begin{bmatrix} a_0 & a_1 & a_2 & \cdots & a_{n_r} & b_1 & b_2 & \cdots & b_{n_r} & c_0 & c_1 & c_2 & \cdots & c_{n\theta} & d_1 & d_2 & \cdots & d_{n\theta} \end{bmatrix} \tag{D.6}$$

To ensure that the first eight coefficients take on the right values, equality constraints have been applied to them, effectively fixing them on the designated value according to the boundary conditions. This did result in a solution, but it was not a feasible one. The trajectory matched the boundary conditions, but its shape and the corresponding thrust profile led to a total ΔV of more than 150 km/s for a single-revolution trajectory from Earth to Mars.

To resolve this problem more constraints have been applied. For a trajectory to the outer Solar System it can be said that both the radius r and the transfer angle θ can only increase. Therefore, it has first been tried with only constraints on r. This led to some sudden directional changes at the end creating a zigzag-pattern that caused the ΔV to go up to about 540 km/s. To correct for this, the constraints on θ were activated as well and finally a correct and feasible trajectory shape was found that only required a ΔV of 6.08 km/s. However, its thrust profile did not match the results obtained by Taheri and Abdelkhalik [21] at all and thus the thrust constraint was turned on too. It did not have a positive outcome, as the best solution still did not resemble the comparison case.

For that reason, it has been decided to redefine the decision vector and exclude the first eight coefficients from the solving process. With this new approach, the decision vector is denoted by Equation (D.7):

$$\mathbf{x} = \begin{bmatrix} a_0 & a_3 & a_4 & \cdots & a_{n_r} & b_3 & b_4 & \cdots & b_{n_r} & c_0 & c_3 & c_4 & \cdots & c_{n\theta} & d_3 & d_4 & \cdots & d_{n\theta} \end{bmatrix} \tag{D.7}$$

This interpretation does not require the implementation of any constraints and is capable of reproducing the same results as were obtained by Taheri and Abdelkhalik [21].

D.3. 2D Unconstrained Finite Fourier Series

When the first test run with the unconstrained finite Fourier series method was done, some peculiar behaviour was observed. The trajectory to Mars has been set up with the exact same settings as in Section 8.1, but this time the thrust acceleration was not limited. The trajectory solution that was obtained is plotted in Figure D.1.

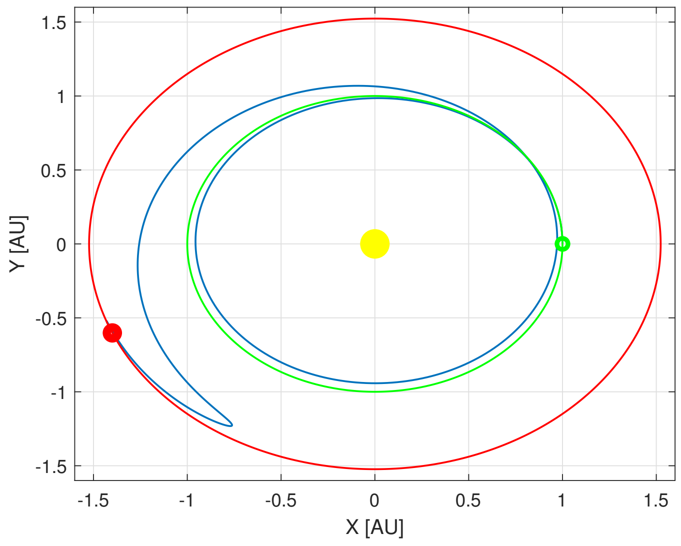


Figure D.1: The result of the first run obtained with the unconstrained finite Fourier series.

The required ΔV for this trajectory is a whopping 125.5 km/s, which is far beyond the nominal value of about 5.6 km/s to 5.8 km/s. The reason why it is so high can be attributed to the sudden change in direction near the end of the trajectory. This directional change requires the spacecraft to kill all its velocity and move into the opposite direction, rendering the solution infeasible.

The problem has been solved by adding additional constraints on the transfer angle. It is prescribed that the transfer angle at DP n should always be larger than the transfer angle at DP $(n-1)$, ensuring an orbit shape that always continues in the same direction. This also means that the name *unconstrained finite Fourier series* only refers to the absence of the thrust constraint. The total number of constraints on the transfer angle depends on the number of discretisation points and can be described as $(\#DP - 1)$.

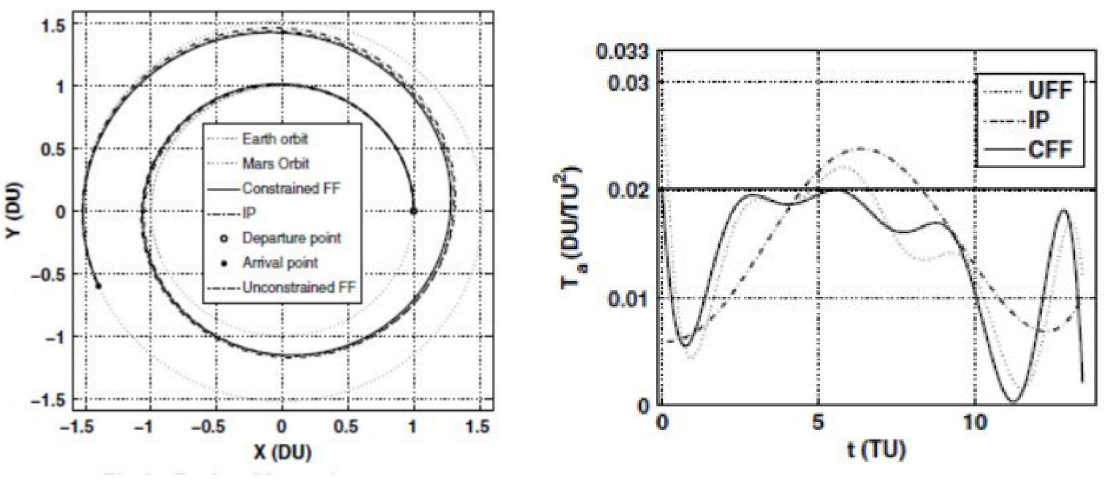
D.4. Validation

The results Taheri and Abdelkhalik [21] show in their paper is based on the following input parameters:

Table D.1: Input parameters and boundary conditions for the trajectory from Earth to Mars [21].

Boundary Conditions		Input Parameters	
r_i	1 DU	N_{rev}	1
θ_i	0 rad	n_r	2
r_f	1.5234 DU	n_θ	5
θ_f	9.831 rad	$T_{a_{max}}$	0.02 DU/TU^2
$\dot{r}_i$	0 DU/TU	# DP	22
$\dot{\theta}_i$	1 rad/TU	TOF	13.447 TU
$\dot{r}_f$	0 DU/TU		
$\dot{\theta}_f$	0.5318 rad/TU		

The graphical results of their interplanetary rendez-vous case are shown in Figure D.2. It contains a figure of the constrained and unconstrained trajectories, as well as their corresponding thrust profiles.



(a) The trajectory from Earth to Mars. (b) The thrust profile for the trajectory from Earth to Mars.

Figure D.2: The trajectory from Earth to Mars and the corresponding thrust profile, based on the data from Table 7.2 [21].

The paper by Taheri and Abdelkhalik [21] does not provide any information on the value of the coefficients that produce this trajectory, therefore the coefficient that the author of this thesis found are enclosed in Table D.2.

Table D.2: The coefficients that are used to compute the trajectories in Figure D.2a.

	a_0	a_1	a_2	b_1	b_2	c_0	c_1	c_2	c_3	c_4	c_5	d_1	d_2	d_3	d_4	d_5
Unconstrained	2.53	-0.26	0	0	0	10.3	-5.56	-0.28	0.67	0.05	-0.03	0.6	2	0.15	-0.18	-0.01
Constrained	2.53	-0.26	0	0	0	9.9	-5.56	-0.03	0.67	0	-0.03	0.96	2	0.01	-0.18	0

D.5. Solving Process

Taheri and Abdelkhalik [21] do not state the exact number of discretisation points that have been used to compute both trajectories, but only claim that at least 15 discretisation points are required to capture the topologies of the trajectory and that after 80 discretisation points no significant improvements are made. When the provided reference case as in Table D.1 is executed, it neatly matches Figure D.2, but if a run over the entire range of DPs is done, the obtained ΔV values show the following pattern as in Figure D.3.

This figure clearly indicates that at some discretisation points the solver does not converge to the right solution. Hence this means that it remains unclear whether this is inherent to the method, or due to the implementation as described in Chapter 5. This phenomenon was eventually resolved by implementing a Monte Carlo component in the simulation and performing a multi-start with 15 different decision vectors which were based on a normal distribution of the first initial guess. The results of this multi-start is shown in Figure D.4. Note that the range has been significantly increased to clearly show the asymptotic behaviour.

D.6. Reference Frame

The second step in setting up a finite Fourier series trajectory model is to define the boundary conditions. The following equation is given in order to compute the transfer angle θ_f at the position of the arrival body (explaining the subscript f) as a function of the number of revolutions N_{rev} [21]:

$$\theta_f = \theta_0 + N_{\text{rev}} \cdot 2\pi \tag{D.8}$$

in which θ_0 represents the initial angle between the departure body at the start ($t = 0$) and the arrival body at the end ($t = T$), measured counter-clockwise. This is nicely illustrated by Figure D.5.

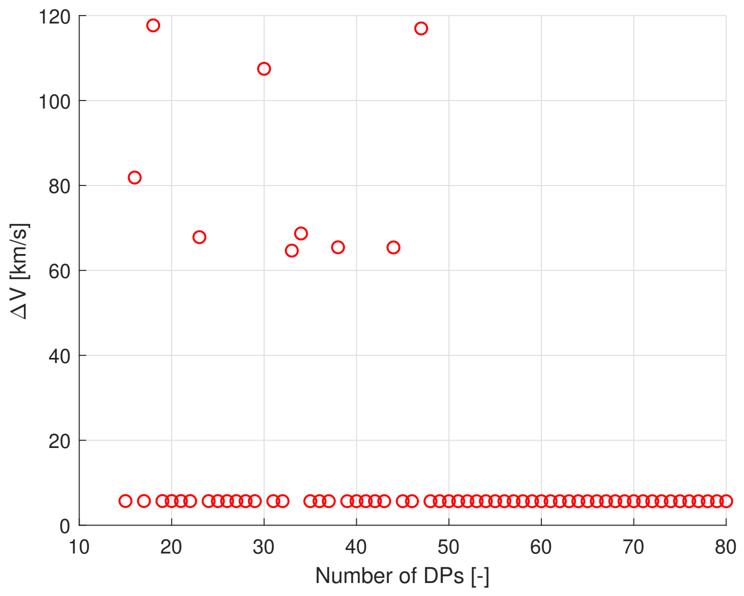


Figure D.3: Excessively high ΔV values are found for some random discretisation points.

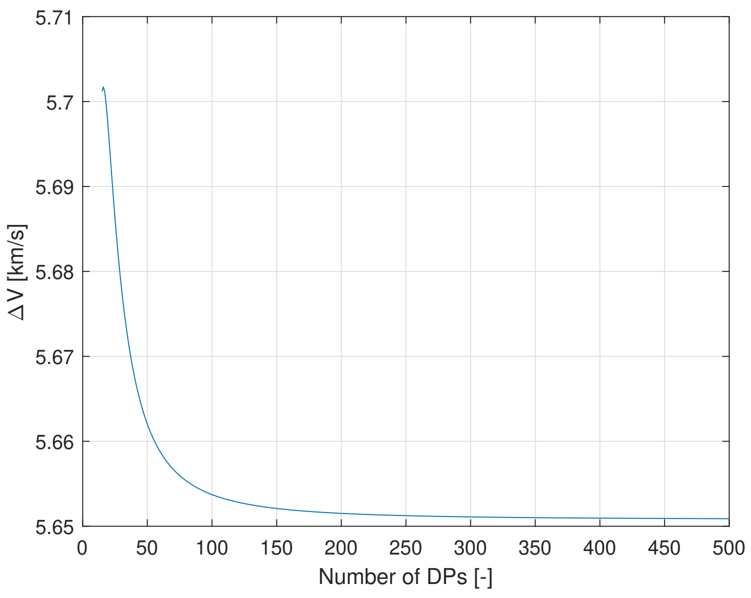


Figure D.4: After the multi-start feature has been implemented, the ΔV behaves as it should for an increasing number of discretisation points.

Taheri and Abdelkhalik [21] however do not mention that the transfer angle is always normalised in such a way that the positive x-axis always serves as the datum line from which θ is measured. This causes problems when the SPICE libraries are used to obtain the state vectors of celestial bodies with respect to the ecliptic reference frame, as this normalisation is not accounted for.

In order to solve it, two things can be done: either the state vector is corrected for this normalisation, or Equation (D.8) is slightly altered. The approach that has been followed in this thesis and thus in the implementation as explained in Chapters 5 and 6 is the latter one, in which the initial transfer angle θ_i is added, resulting in Equation (D.9).

$$\theta_f = \theta_0 + N_{\text{rev}} \cdot 2\pi + \theta_i \tag{D.9}$$

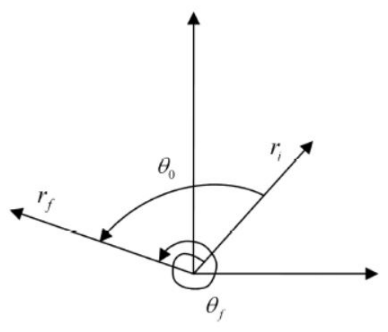
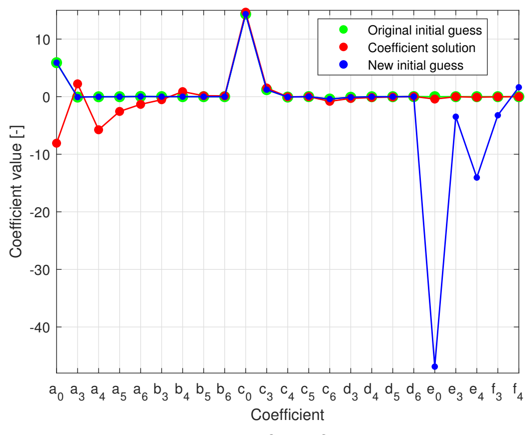


Figure D.5: A graphical representation explaining how the initial angle θ_0 between the departure body at $t = 0$ and the arrival body at $t = T$ is found.

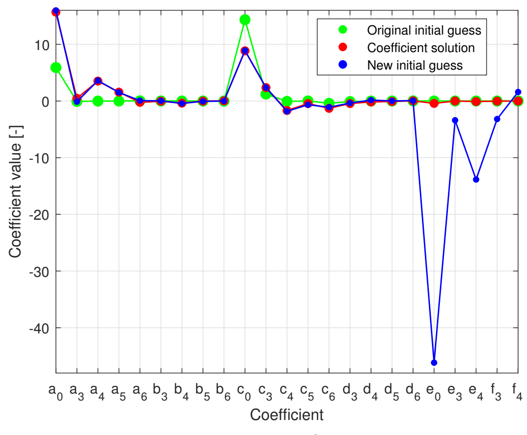
E
Fourier Coefficients

As an addition to Section 11.4, the figures that have not been discussed separately are included below as well. Figures E.1 and E.3 show the results of the trajectory to Jupiter and Dionysus respectively while the settings for case II have been applied: the coefficients for r, θ and z are all estimated by the initialisation function that can be read in the title of the subfigures.

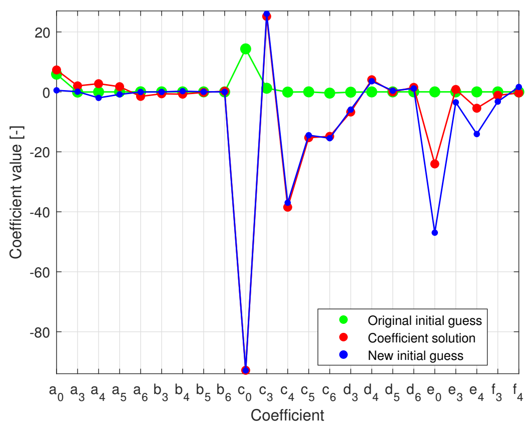
Furthermore, Figures E.2 and E.4 contain the outcome of the trajectory solutions to Jupiter and Dionysus respectively while the settings for case III have been applied: the coefficients for r and θ are estimated by the functions that are written in the title of the subfigures and the coefficients for z are set equal to zero (left), or the coefficients for both r, θ and z are approximated by the initialisation function in the title of the subfigures (right).



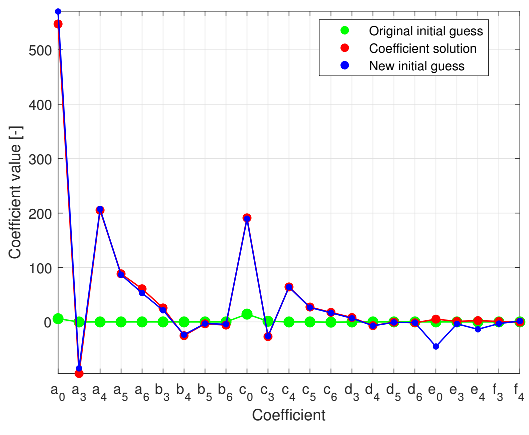
(a) $f(\tau) = a\tau^3 + b\tau^2 + c\tau + d$



(b) $f(\tau) = ae^{b\tau} + c$

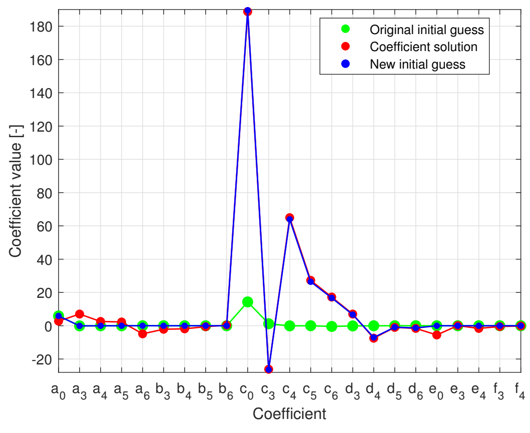


(c) $f(\tau) = A\sin\left(\omega\left(\tau - \phi\right)\right) + K$

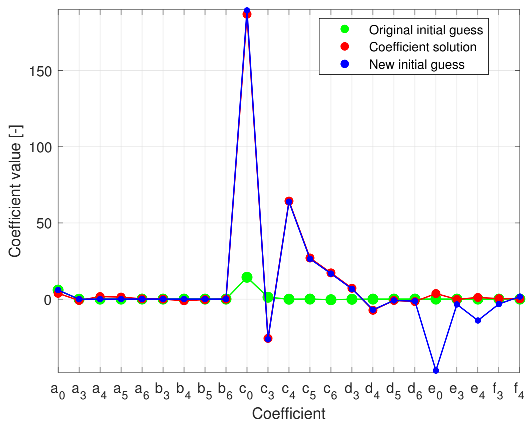


(d) $f(\tau) = a + b\ln(\tau)$

Figure E.1: The newly obtained Fourier coefficients and their (original) initial guess for a trajectory to Jupiter. Following the outline of case II the mentioned function that serves as a subtitle is used to approximate both r, θ and z.

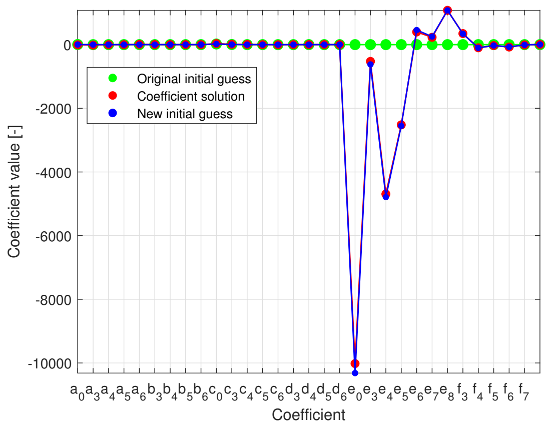


(a) The Fourier coefficients where r is approximated by a power function and θ by a logarithmic function, while z is set to zero.

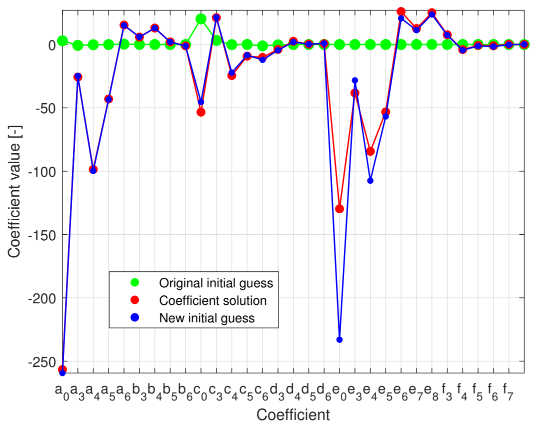


(b) The Fourier coefficients where r is approximated by a power function and θ by a logarithmic function, while z is approximated by a sinusoidal function.

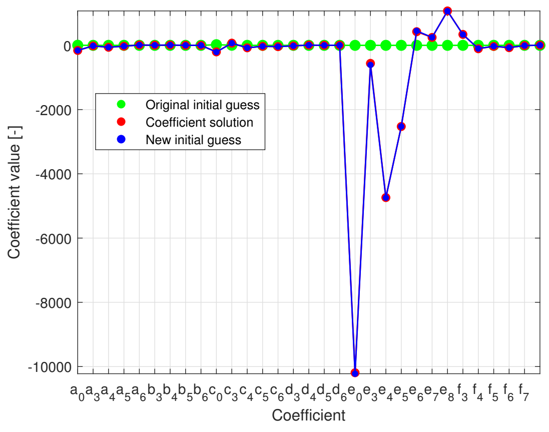
Figure E.2: The newly obtained Fourier coefficients and their (original) initial guess for a trajectory to Jupiter.



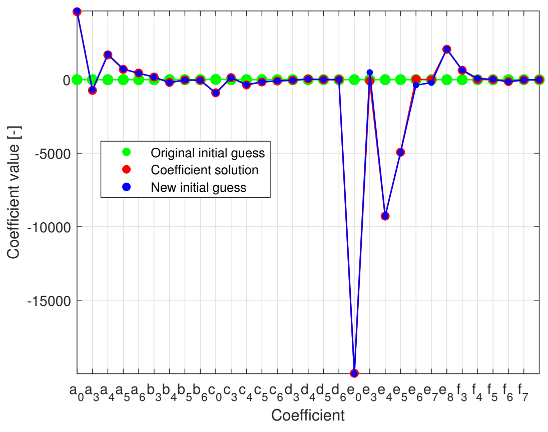
(a) $f(\tau) = a\tau^3 + b\tau^2 + c\tau + d$



(b) $f(\tau) = ae^{b\tau} + c$

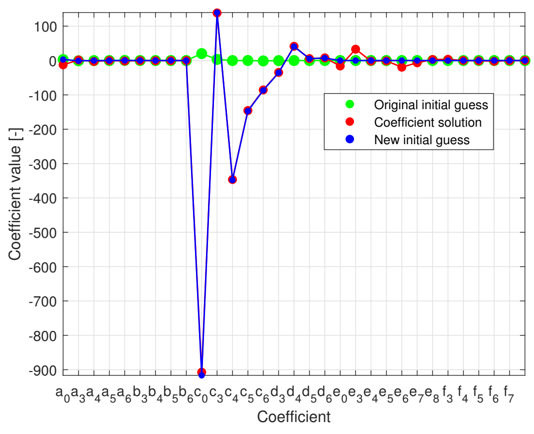


(c) $f(\tau) = A\sin\left(\omega\left(\tau-\phi\right)\right) + K$

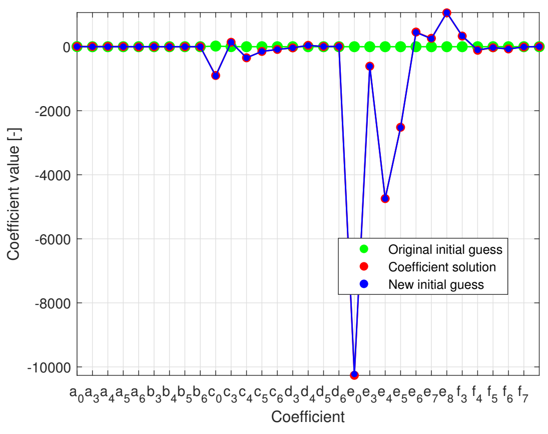


(d) $f(\tau) = a + b\ln(\tau)$

Figure E.3: The newly obtained Fourier coefficients and their (original) initial guess for a trajectory to Dionysus. Following the outline of case II the mentioned function that serves as a subtitle is used to approximate both r, θ and z.



(a) The Fourier coefficients where r is approximated by a power function and θ by a logarithmic function, while z is set to zero.



(b) The Fourier coefficients where r is approximated by a power function and θ by a logarithmic function, while z is approximated by a sinusoidal function.

Figure E.4: The newly obtained Fourier coefficients and their (original) initial guess for a trajectory to Dionysus.

F Background Information on Shape-Based Methods

The methods are arranged chronologically with respect to the year of the first publication, starting with the exponential sinusoid in Appendix F.1, followed by the inverse polynomial in Appendix F.2 and the spherical shaping method in Appendix F.3. At last, the hodographic shaping method is worked out in Appendix F.4, after which the finite polynomial method is elaborated upon in Appendix F.5.

F.1. Exponential Sinusoid

The exponential sinusoid (exposin in short) is the very first revolutionary method to be used for low-thrust trajectory design. It is designed by Petropoulos and Longuski [14] and requires the equation of motion as given in Equation (2.1) to be rewritten in polar coordinates, which results in Equation (2.2) and is here stated for the sake of clarity:

$$\begin{cases} \ddot{r} - r\dot{\theta}^2 + \frac{\mu}{r^2} = T_a \sin\alpha \\ 2\dot{r}\dot{\theta} + r\ddot{\theta} = T_a \cos\alpha \end{cases} \tag{F.1}$$

The exponential sinusoid is then described as follows:

$$r = k_0 \mathrm{e}^{(k_1 \sin(k_2\theta + \phi))} \tag{F.2}$$

In here, k_0, k_1, k_2 and ϕ are constants. k_0 Represents a scaling factor. k_1 Is the dynamic range parameter that influences the ratio of the apoapsis distance to the periapsis distance. The winding parameter k_2 is a measure for the number of revolutions the trajectory will cover. This effect is depicted in Figure F.1. The phase angle ϕ determines the orientation of the spiral in the plane. The method makes use of tangential thrust. This means that the thrust vector is always pointing in the direction of the velocity vector or against it.

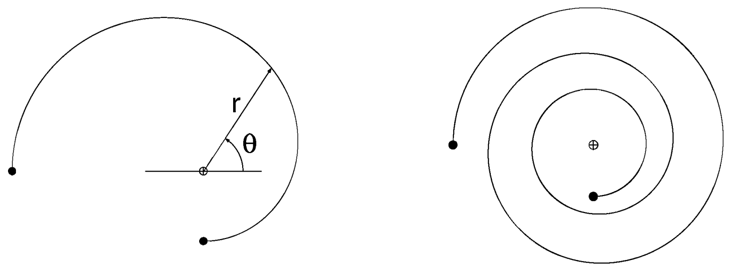


Figure F.1: The effects of the winding parameter k_2 [14]. In the left spiral $k_2 = 2/3$, while at the right spiral it is equal to $2/11$.

By inserting the shape function of Equation (F.2) into the polar equations of motion in Equation (F.1) and rewriting it, an analytic relation for the angular rate $\dot{\theta}$ is found.

$$\dot{\theta}^2 = \left(\frac{\mu}{r^3}\right)\frac{1}{\tan^2(\gamma) + k_1 k_2^2 \sin(k_2\theta + \phi) + 1} \tag{F.3}$$

Finally, the thrust angle α is defined in Equation (F.4). Tangential thrust is chosen, because it is the simplest analytic case and results in an attractive decoupling of the angular velocity and acceleration. With tangential thrust, the thrust angle α is equal to the flight path angle γ. In case retrograde thrust is applied, the thrust vector is pointing in the exact opposite direction. A value of n equal to 0 therefore represents prograde thrust, while the spacecraft experiences retrograde thrust in case n is equal to 1.

$$\alpha = \gamma + n\pi \tag{F.4}$$

The time of flight (TOF) for this method can be found by integrating the inverse of the radial velocity from 0 to the final angle θ_f, as seen in Equation (F.5) [6]. Any possible out-of-plane motion is not taken into account for the TOF calculation.

$$\text{TOF} = \int_0^{t_f} \mathrm{d}t = \int_0^{\theta_f} \frac{\mathrm{d}t}{\mathrm{d}\theta}\mathrm{d}\theta \tag{F.5}$$

The total ΔV that is required to fly the mission can be computed from the integration of the thrust acceleration T_a, as defined in Equation (2.2), over the transfer angle [6].

$$\Delta V = \int_0^{t_f} T_a\,\mathrm{d}t = \int_0^{\theta_f} \frac{T_a}{\dot{\theta}}\mathrm{d}\theta \tag{F.6}$$

3D Motion

For modelling three-dimensional motion, the exposin uses an additional thrust acceleration f_h, which acts perpendicular to the angular momentum vector of the spacecraft. It is defined as follows:

$$f_h = \frac{\mu}{r^2}\left[a_{0_P} + b_0\frac{r_{min}}{r}\right] \tag{F.7}$$

in which a_{0_P} and b_0 are constants and r_{min} is the periapsis radius of the trajectory. The scaling of the equation with $\frac{1}{r^2}$ is done such that f_h is adjusted according to the degradation of received solar power. The f_h component is shaped such that it matches the three-dimensional position at final time.

Discussion

The exponential sinusoid is a very simple and fast shape-based method with only four shape parameters that can also be combined with gravity assists. As this is one of the very first shape-based methods, it has been revolutionary in the field of preliminary trajectory design.

Nonetheless, it lacks in terms of efficiency and the ability of generating feasible trajectories, because it does not have a boundary condition set on its velocity, which thus has to be obtained through iterations, and its thrust profile can only be acquired a posteriori, which could yield excessively higher thrust levels than that are currently technologically feasible. Also, a three-dimensional trajectory is only approximated.

Pro:
- It quickly describes the trajectory with four shape parameters.
- It can be combined with gravity assists.

Con:
- The out-of-plane motion is only approximated.
- Only tangential thrust can be applied.
- It does not have any boundary conditions on velocity.
- The thrust profile is determined a posteriori.

F.2. Inverse Polynomial

When designing a new shape-based method, Wall and Conway [28] tried to extend Petropoulos' exposin with two additional free parameters. Unfortunately, this did not work out, so then a curve-fitting algorithm was applied to a known trajectory. From here it was found that the orbit was best represented by a fifth or sixth-order inverse polynomial (IP):

$$r(\theta) = \frac{1}{a + b\theta + c\theta^2 + d\theta^3 + e\theta^4 + f\theta^5 + g\theta^6} \tag{F.8}$$

Depending on whether the transfer time will be specified or not, a sixth or fifth-order polynomial will be used, respectively. For this method it is important to find the values of the constants a, b, c, d, e, f, and g. In the following explanation, first the solution of the fifth-order polynomial will be explained.

As the inverse polynomial method defines the thrust in the same way as the previously discussed exposin method, Equation (F.4) applies as well. The constraints for the initial and final flight path angle need to be defined. To do this, the time derivative of Equation (F.8) needs to be found. With this derivative, the flight path angle γ is given by the following expression (recall Figure 2.4):

$$\tan(\gamma) = \frac{\dot{r}}{r\dot{\theta}} = -r \cdot \left(b + 2c\theta + 3d\theta^2 + 4e\theta^3 + 5f\theta^4\right) \tag{F.9}$$

Filling in the boundary conditions for the start and the end point of the trajectory, i.e. $\theta_0 = 0$ and $\theta_f = \theta_f$, the two flight path angle constraints can be determined, leading to:

$$\tan(\gamma_1) = -r_1 \cdot b \tag{F.10a}$$

$$\tan(\gamma_2) = -r_2 \cdot \left(b + 2c\theta_f + 3d\theta_f^2 + 4e\theta_f^3 + 5f\theta_f^4\right) \tag{F.10b}$$

To find the other polynomial coefficients, Equation (F.8) is to be plugged into the equations of motion in Equation (2.2). If then the assumption for the thrust direction as described in Equation (F.4) is implemented, the expression can be rewritten for $\dot{\theta}$:

$$\dot{\theta}^2 = \frac{\mu}{r^4} \frac{1}{\left[(1/r) + 2c + 6d\theta + 12e\theta^2 + 20f\theta^3\right]} \tag{F.11}$$

Whereas Equations (F.10a) and (F.10b) guarantee that the velocity direction (i.e. the flight path angle) of the spacecraft matches the initial and final orbits, Equation (F.11) ensures that the velocity magnitude is correct as well.

The first three coefficients a, b an c can be obtained by filling in the initial conditions in the previously found equations. The other three, i.e. d, e and f, can be found by feeding the terminal BCs into those equations and then solving a linear system of equations.

When all coefficients are known, the TOF can be computed by integrating Equation (F.11) over the total transfer angle, which Wall and Conway [28] did by means of a root-finding algorithm. This time can however not be used to calculate the position of the target body at r_2, as it is not constrained. Therefore, it is recommended to use the fifth-order polynomial in case of time-free problems, as the point of rendez-vous is not fixed here, hence θ_f is free. However, in case it is desired to specify the transfer time, the sixth-order inverse polynomial must be used. For this version of the shape function, the coefficients a, b and c are still computed in the same way. The new transfer time constraint will be represented by parameter d, for it has the smallest sensitivity to change in transfer time, which is beneficial to root-finding algorithms. The other four coefficients are again determined by solving a linear system of equations.

It is important to note that the expression for $\dot{\theta}$ in Equation (F.11) changes slightly by adding the g coefficient:

$$\dot{\theta}^2 = \frac{\mu}{r^4} \frac{1}{\left[(1/r) + 2c + 6d\theta + 12e\theta^2 + 20f\theta^3 + 30g\theta^4\right]} \tag{F.12}$$

To find coefficient d, this equation is integrated over the total transfer angle θ_f and equated to the required TOF. With a root-finding algorithm the required value for d can be determined, such that the inverse polynomial shape function complies with the constraint on the transfer time.

To eventually determine the thrust acceleration T_a as defined by Equation (2.2), the only unknown variable in this equation is the second derivative of θ. This is simply determined by taking the derivative of Equation (F.12) (or Equation (F.11) in case of a fifth-order polynomial). Plugging this in the EoM results in the following expression for the thrust acceleration:

$$T_a = -\frac{\mu}{2r^3\cos(\gamma)} \frac{6d + 24e\theta + 60f\theta^2 + 120g\theta^3 - (\tan(\gamma)/r)}{\left[(1/r) + 2c + 6d\theta + 12e\theta^2 + 20f\theta^3 + 30g\theta^4\right]^2} \tag{F.13}$$

Finally, the total velocity increment for the entire trajectory is found by:

$$\Delta V = \int_0^{\theta_f} \frac{T_a(\theta)}{\dot{\theta}(\theta)} \mathrm{d}\theta \tag{F.14}$$

which is an interesting result, as the exhaust velocity does not seem to influence the required ΔV. It is purely based on the shaping function.

3D Motion

In the extension of the inverse polynomial method to three-dimensional space, Wall [27] simply added a third component to the polar equations of motion, transforming them to a cylindrical coordinate system, as specified by Equation (2.3). This shaping function of the third dimension is a polynomial too:

$$z(\theta) = a_z + b_z\theta + c_z\theta^{q-1} + d_z\theta^q \tag{F.15}$$

in which a_z, b_z, c_z and d_z are the polynomial coefficients, while q denotes the highest order of the polynomial. With the initial and final positions and velocities in the z-direction, the four coefficients can be solved. At the initial position, θ is equal to zero, which means the coefficients a_z and b_z can be determined directly. The other two coefficients are found by solving another matrix equation.
Once these coefficients are known, the thrust acceleration component in the z-direction, T_{a_z} can be determined from the third equation in Equation (2.3), which results in:

$$T_{a_z} = \frac{\mu}{r^3}z - \left[c_z(q-1)(q-2)\theta^{q-3} + d_z q(q-1)\theta^{q-2}\right]\dot{\theta}^2 + \left[b_z + c_z(q-1)\theta^{q-2} + d_z q\theta^{q-1}\right]\ddot{\theta} \tag{F.16}$$

The total thrust acceleration is then found by the norm of the in-plane thrust acceleration $T_{a_{r\theta}}$ and the out-of-plane thrust acceleration T_{a_z}:

$$T_a = \sqrt{T_{a_{r\theta}}^2 + T_{a_z}^2} \tag{F.17}$$

Discussion

The main purpose of the sixth-order inverse polynomial is its applicability to rendez-vous problems in fixed time. It has however been shown that it is effective in computing interception trajectories and orbit transfers as well. The fifth-degree variant can be used for these same problems in case the transfer time is not specified. It also demonstrated to work well with a genetic algorithm (GA).

Furthermore, in case of a rendez-vous trajectory, the inverse polynomial method proved to be able to successfully create near-optimal trajectories. In the first run of an asteroid rendez-vous problem Wall and Conway [28] optimised for spacecraft mass. When this shape-based solution was later on used as an initial guess for a more accurate solver, the final mass only differed by a few percent [28].

With the extension of the method into the third dimension, a wider spectrum of trajectories can be computed. Wall [27] however mentions that this approach is rather limited, as it can only handle inclinations of up to 15°. Up to a plane change of this degree, the assumption that z is small relative to r holds, which still means that missions towards most celestial objects can be designed properly. As soon as higher inclinations are to be achieved, the method becomes too inaccurate.

Pro:

- It is simple to use.
- It is applicable to rendez-vous and interception trajectories, and orbit raising problems.

- It works well with a genetic algorithm optimiser.

Con:
- Its 3D performance is below average.
- The thrust profile is fully determined by the BCs.
- The TOF is found iteratively.

F.3. Spherical Shaping Method

After the exposin and the inverse polynomial, it were Novak and Vasile [12] who came up with an even more advanced method: the spherical shaping method. This method is based on a three-dimensional description of the state of the spacecraft in spherical coordinates. The spherical shaping method differs from the others that have been described in this chapter so far, as the parameters that describe the trajectory are not parametrised by time t, but by the azimuthal angle θ. This gives the following state vector with $r = R(\theta)$, $t = T(\theta)$ and $\phi = \Phi(\theta)$:

$$\tilde{\mathbf{x}} = \begin{bmatrix} r & t & \phi & \frac{\mathrm{d}R}{\mathrm{d}\theta} & \frac{\mathrm{d}T}{\mathrm{d}\theta} & \frac{\mathrm{d}\Phi}{\mathrm{d}\theta} \end{bmatrix}^T \tag{F.18}$$

The assumed shapes are defined with the three formulas for $R(\theta)$, $T(\theta)$ and $\Phi(\theta)$. The coefficients in those functions will be dependent on the boundary conditions of the trajectory, i.e. position and velocity. The convenience of these functions is that they can be solved analytically, resulting in the thrust profile, the ΔV-budget and the spacecraft mass, all of which are crucial design parameters.

To analyse the motion directly, Equation (2.1) needs to be rewritten to account for the parametrisation by θ. Substituting the newly parametrised equations of motion in Equation (2.1) results in:

$$\dot{\theta}^2\mathbf{r}'' + \ddot{\theta}\mathbf{r}' = -\mu\frac{\mathbf{r}}{r^3} + \mathbf{u} \tag{F.19}$$

in which $\mathbf{u}$ corresponds to the thrust acceleration vector $\mathbf{T_a}$, which is defined as the total thrust T divided by the spacecraft mass m. It should also be noted that the prime notation indicates the derivative with respect to θ.

The trajectory shape is defined through shaping functions that specify the radial distance R and the elevation angle Φ. The only conditions that must hold, is that R can only be positive, and Φ should remain on the interval $\left[-\frac{\pi}{2}, \frac{\pi}{2}\right]$.

Both Novak and Vasile [12] and Roegiers [17] suggest that the following two functions are to be used as shaping functions. The radius is shaped through an inverse polynomial, while the elevation angle is a polynomial equation. These equations were chosen as the boundary conditions on the position and velocity can be obtained analytically.

$$\begin{cases} R(\theta) & = & \dfrac{1}{a_0 + a_1\theta + a_2\theta^2 + (a_3 + a_4\theta)\cos(\theta) + (a_5 + a_6\theta)\sin(\theta)} \\ \Phi(\theta) & = & (b_0 + b_1\theta)\cos(\theta) + (b_2 + b_3)\sin(\theta) \end{cases} \tag{F.20}$$

These functions are however not strictly bounded to the spherical shaping method, which makes it rather a shape-based framework than a method. In his MSc thesis, Vroom [25] proposed another shaping function for the elevation angle, because the three-dimensional performance of the method could be improved. His shaping function is shown below in Equation (F.21):

$$\Phi(\theta) = \sqrt{\frac{1+\left(a_0 \mathrm{e}^{a_1\theta+a_2} + a_3\mathrm{e}^{a_4\theta+a_5}\right)}{1+\left(a_0 \mathrm{e}^{a_1\theta+a_2} + a_3\mathrm{e}^{a_4\theta+a_5}\right)\cos^2\left(p_0\theta^{p_1}+p_2\theta+p_3\right)}}\cos\left(p_0\theta^{p_1} + p_2\theta + p_3\right)\left(b_0 + b_1\theta + b_2\theta^2\right) + c \tag{F.21}$$

This new shaping function contains already 13 coefficients. Ten of them can be covered by the boundary conditions, but the others will have to be found by means of an optimisation process, and thus it will be more computationally intensive.

The only shape function that is yet to be determined is $T(\theta)$. Novak and Vasile [12] defined the expression for T through its derivative:

$$T' = \sqrt{\frac{D(\theta) R(\theta)^2}{\mu}} \tag{F.22}$$

In here, $D(\theta)$ is the time equation scalar function. An interesting property of D is that it is not dependent on the reference frame. It solely depends on the shape of the trajectory. As Equation (F.22) is a root function, real values are only found for T' in case D is positive, meaning the curvature is towards the central body. Figure F.2 shows the effects of the curvature on the flown trajectory.

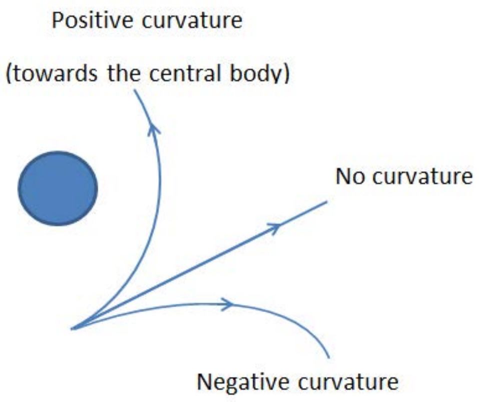


Figure F.2: A graphical depiction of the curvature depending on the time equation scalar function D [17].

When the normal component of the thrust acceleration is substituted by the radius vector **r**, expressed in spherical coordinates, as seen in Equation (2.6), the following formula can be used for $D(\theta)$:

$$D = -r'' + 2\frac{r'^2}{r} + r'\phi'\frac{\phi'' - \sin(\phi)\cos(\phi)}{\phi'^2 + \cos^2(\phi)} + r\left(\phi'^2 + \cos^2(\phi)\right) \tag{F.23}$$

The velocity and acceleration vector can be computed, as well as the control acceleration to find the necessary ΔV and the final mass of the spacecraft. The total ΔV is calculated according to:

$$\Delta V = \int_0^{t_f} \mathbf{u}\, \mathrm{d}t = \int_{\theta_0}^{\theta_f} \mathbf{u} \left|\frac{\mathrm{d}t}{\mathrm{d}\theta}\right| \mathrm{d}\theta = \int_0^{\theta_f} \mathbf{u} |T(\theta)|\, \mathrm{d}\theta \tag{F.24}$$

while the total time of flight is found by:

$$TOF = \int_0^{t_f} \mathrm{d}t = \int_0^{\theta_f} |T(\theta)|\, \mathrm{d}\theta \tag{F.25}$$

Discussion

The spherical shaping method designed by Novak and Vasile [12] can be used in three dimensions and returns much better results with respect to inclined orbits than the average shape-based method. With the improvements made by Roegiers [17] and Vroom [25] quite some more research has been done on this method which led to an improvement from a maximum achievable inclination of 50° up to 70°.

This immediately leads to another asset of the method: it can be used with various shaping functions. The spherical shaping method could be regarded as a shape-based framework, rather than a method, which offers a large amount of freedom compared to the previously discussed shape-based methods.

The first major drawback is found in the TOF constraint. This cannot be computed analytically and has to be determined through an iterative process, which reduces the speed advantage shape-based methods usually have over regular trajectory design tools.

Moreover, the improvement for inclined orbits came together with a decline in the quality of the computed ΔVs, as these did not yield correct results.

Finally, the method is not capable of having a thrust constraint implemented in the shaping functions, which translates to many trajectories that are unfortunately not feasible, due to limitations in achievable engine technology.

Pro:
- It performs relatively well in case of three-dimensional orbits.
- It can be used with various shaping functions.

Con:
- In case more than ten coefficients are used for the shaping functions, the TOF constraint needs to be satisfied by an iterative process.
- With the new shaping function for the elevation angle by Vroom [25], the calculated ΔV are off.
- The thrust is found a posteriori and cannot be constrained beforehand.

F.4. Hodographic Shaping Method

Another method which TU Delft significantly contributed to is the hodographic shaping method by Gondelach [6]. In fact, it was designed at TU Delft. It uses a different approach than all other methods that are discussed in this chapter. Often it is the radius or the polar angle of the trajectory that is shaped, but this does not hold for the hodographic shaping method. In this method the perspective has been shifted to the velocity domain, hence the shaping functions shape the velocity of the transfer trajectory. The name arises from velocity hodographs, as seen in Figure F.3. The path in the velocity domain can be integrated over time to obtain a path in the spatial domain.

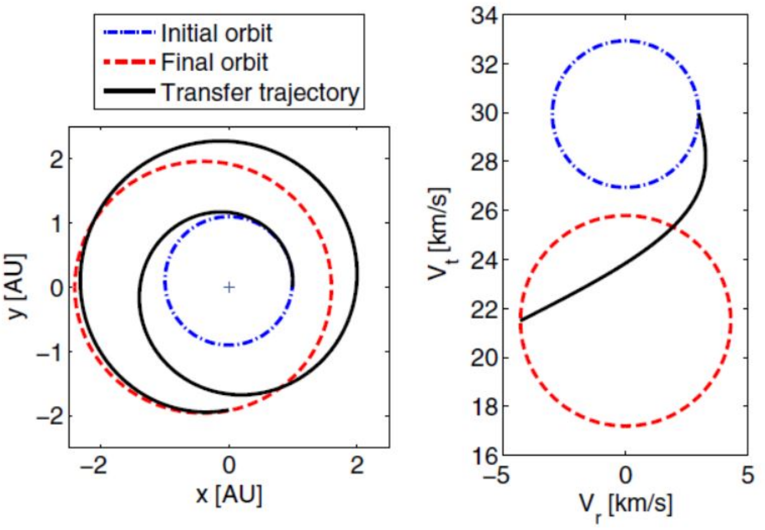


Figure F.3: A transfer trajectory with its corresponding velocity hodograph [5].

The method has the capabilities to work in three dimensions and thus it uses the cylindrical coordinate system, as specified in Equation (2.3). The hodographs also allow for two approaches to be taken: time- or polar angle-driven. First the time-driven case will be explained, followed by the polar angle-driven elaboration.

In the time-driven case, the three velocity components in the radial direction (V_r), the transverse direction (V_θ) and the axial direction (V_z) are shaped as a function of t. The radial and axial distances are then found by analytically integrating their corresponding velocity functions according to:

$$r(t) = r_0 + \int_0^t V_r \, \mathrm{d}t \tag{F.26a}$$

$$z(t) = z_0 + \int_0^t V_z \, \mathrm{d}t \tag{F.26b}$$

The polar angle θ is extracted by the integration of the angular velocity:

$$\theta(t) = \theta_0 + \int_0^t \dot{\theta}\,\mathrm{d}t = \theta_0 + \int_0^t \frac{V_\theta}{r}\,\mathrm{d}t \tag{F.27}$$

The various thrust components can be computed by substituting the three velocity functions in Equation (2.3). The total thrust acceleration is then given by:

$$T_a = \sqrt{T_{a_{r\theta}}^2 + T_{a_z}^2} \tag{F.28}$$

Finally, the ΔV that is required for the trajectory to be flown is obtained by simply integrating the thrust over the time of flight:

$$\Delta V = \int_0^{t_f} T_a\,\mathrm{d}t \tag{F.29}$$

The base functions to describe the velocity can take various forms, as long as they are analytically solvable to minimise the computational effort. In this way, trigonometric, polynomial or exponential terms can serve as a base function, which implies that each velocity function can be described by a summation of base functions $v_i(t)$, multiplied by their corresponding coefficient c_i:

$$V(t) = \sum_{i=1}^{n} c_i v_i(t) \tag{F.30}$$

where the minimum number of base functions is of course dependent on the number of boundary conditions. As each velocity function will have to satisfy three boundary conditions, there are nine equations to be solved. This leads to a matrix equation that finds expressions for the coefficients c_1 through c_3. This holds for the radial and axial velocities. The transverse component is found in a slightly different but similar way, as the polar angle θ is found by integrating the angular velocity $\dot{\theta}$.

Taking a closer look at the polar angle-driven method, many similarities with the spherical shaping method described in Appendix F.3 have been found. Again, there will be a shaping function for the radius $R(\theta)$, a shaping function for the z-component $Z(\theta)$, and ultimately there is the time evolution function $T(\theta)$. The reason why there is not a function for θ itself, has to do with the fact that it is θ which is the independent variable. Therefore, the time evolution function exists, which is basically the inverse of $\dot{\theta}$. The shaping functions are shown below. Note the difference in notation with respect to the spherical shaping method, as these functions below would correspond to $R'(\theta)$, $T'(\theta)$ and $Z'(\theta)$ for the spherical shaping method.

$$r' = \frac{\mathrm{d}r}{\mathrm{d}\theta} = R(\theta)$$

$$t' = \frac{\mathrm{d}t}{\mathrm{d}\theta} = T(\theta)$$

$$Z' = \frac{\mathrm{d}z}{\mathrm{d}\theta} = Z(\theta) \tag{F.31}$$

Just as for the position in r and z as specified by Equations (F.26a) and (F.26b) with the parametrisation by t, a similar approach is followed in the polar angle-driven case:

$$r(\theta) = r_0 + \int_0^t R(\theta)\,\mathrm{d}\theta \tag{F.32a}$$

$$z(\theta) = z_0 + \int_0^t Z(\theta)\,\mathrm{d}\theta \tag{F.32b}$$

The equations to compute the ΔV and the time of flight are exactly the same as for the spherical shaping method, but they are stated here again for the sake of completeness:

$$\Delta V = \int_0^{t_f} \mathbf{T_a}\,\mathrm{d}t = \int_{\theta_0}^{\theta_f} \mathbf{T_a}\left|\frac{\mathrm{d}t}{\mathrm{d}\theta}\right|\,\mathrm{d}\theta = \int_0^{\theta_f} \mathbf{T_a}\,|T(\theta)|\,\mathrm{d}\theta \tag{F.33a}$$

$$TOF = \int_0^{t_f} \mathrm{d}t = \int_0^{\theta_f} |T(\theta)|\ \mathrm{d}\theta \tag{F.33b}$$

To solve for the coefficients that accompany the base functions, exactly the same concept as for the time-driven case can be used.

Discussion

Regarding computation speed, this shaping method is very promising. This has two reasons: the first one being the need of only solving nine boundary conditions, which is a low number compared to the other methods discussed in this chapter. The second reason is in the process of dealing with the boundary conditions. Because they are solved immediately, i.e. without an iterative process, this saves time significantly. Besides, this method does not have any problems when three-dimensional trajectories are to be computed.

The method however does not seem to cope well with multi-revolution trajectories. In case a trajectory would contain more than two full revolutions, the hodographic shaping method is not able to generate good trajectories anymore.

Pro:
- There are no iterative computations needed.
- The method works well in three dimensions.
- The thrust can be pointed in any direction.

Con:
- The performance decreases if the number of revolutions is larger than two.
- The method does not have a thrust magnitude constraint incorporated.

F.5. Finite Polynomial

One of the more recently developed methods is the finite polynomial method [31]. It is quite similar to the inverse polynomial method explained in Appendix F.2. With this method, it is not one shape that is assumed, but a batch of coefficients that assume an approximation of the trajectory. It can be used in three-dimensional motion, is compatible with a thrust constraint and it uses a cylindrical coordinate system. The equations of motion are defined as in Equation (2.3) and s is obtained by using the Pythagorean theorem on the magnitude of the radius vector r and the axial distance vector z. The in-plane motion, i.e. in the xy−plane, is approximated by r and θ which are both a function of time t:

$$\begin{cases} r(t) = a_{r_0} + a_{r_1} t + a_{r_2} t^2 + a_{r_3} t^3 + \ldots + a_{r_m} t^m & \text{(F.34a)} \\ \theta(t) = a_{\theta_0} + a_{\theta_1} t + a_{\theta_2} t^2 + a_{\theta_3} t^3 + \ldots + \ldots + a_{\theta_n} t^n & \text{(F.34b)} \end{cases}$$

In this parametrisation a_{r_0}, a_{r_1} up to a_{r_m} and a_{θ_0}, a_{θ_1} up to a_{θ_n} are the coefficients of the polynomial, which can also be referred to as design parameters. They should be defined such that both m and n are always larger than or equal to three. To rewrite the set of equations in Equation (2.3) in one single expression, some substitutions need to be made, which finally result in:

$$r^2\left(\frac{\mu\dot{\theta}}{s^3} - \dot{\theta}^3\right) + r\left(\dot{\theta}\ddot{r} - \dot{r}\ddot{\theta}\right) - 2\dot{r}^2\dot{\theta} = 0 \tag{F.35}$$

To also describe the out-of-plane motion, a function for z is introduced as well:

$$z(\theta) = a_z\cos(\theta) + b_z\theta + c_z\theta^{q-1} + d_z\theta^q \tag{F.36}$$

where q is a positive integer larger than three and the coefficients a_z, b_z, c_z and d_z are computed from the boundary conditions. The thrust accerlation T_{a_z} is obtained by substituting this equation in the third term of Equation (2.3), resulting in:

$$\begin{aligned} T_{a_z} = \frac{\mu}{s^3}z + \ddot{\theta}\left[-a_z\sin(\theta) + b_z + (q-1)c_z\theta^{q-2} + qd_z\theta^{q-1}\right] + \\ \dot{\theta}^2\left[-a_z\cos(\theta) + (q-1)(q-2)c_z\theta^{q-3} + q(q-1)d_z\theta^{q-2}\right] \end{aligned} \tag{F.37}$$

The total thrust is then computed as the norm of the in-plane thrust acceleration $T_{a_{r\theta}}$ and the out-of-plane thrust acceleration T_{a_z} according to:

$$T_a = \sqrt{T_{a_{r\theta}}^2 + T_{a_z}^2} \tag{F.38}$$

When all required equations are known, the design parameters can be computed. To do so, the shape functions for r, θ and z need to be substituted in Equation (F.35). The boundary conditions are used to compute all unknown coefficients.

The initial design parameters can be found by using the fact that $t_0 = 0$. This allows for the first four coefficients to be computed. The other coefficients are obtained by solving two linear systems of equations for both r and θ, which are in the form of $A\mathbf{x} = B$. For the out-of-plane motion, i.e. in the z-direction, a similar approach is followed.

To compute the other polynomial coefficients, Zeng et al. [31] propose to use the sequential quadratic programming (SQP) approach. The solving procedure can thence be described as follows. First, a reference trajectory is computed based on the eight initial and final boundary conditions from Equation (F.35). The next step is to divide the total time of flight into a set number of discretisation points. Zeng et al. [31] state that by trial and error, they found out that 12 points per revolution is a safe choice. With the previously obtained reference trajectory the state parameters can be calculated at each discretisation point. The outcome should be substituted in Equation (F.38), keeping the constraint into account. The newly acquired set of equations can be solved by means of an SQP approach.

To start this process, an initial guess for the polynomial coefficients is used: the first four coefficients of r and θ are found by approximating the entire trajectory by a cubic polynomial. If the equations that construct the finite polynomial method are then equated to the outcome of the cubic polynomial approximation, an initial guess for the finite polynomial can be found. All eight coefficients of the two cubic polynomials can be computed from the boundary conditions. Once these are known, a simple linear system of the form $A\mathbf{x} = B$ can be set up, in which A contains the finite polynomial values, $\mathbf{x}$ contains the finite polynomial coefficients and B contains the outcome of the cubic polynomial approximation. A more detailed explanation of this procedure can be found in Section 4.3.

Discussion

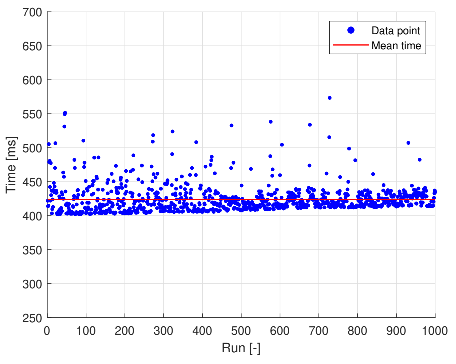
As this method could be regarded as an improvement with respect to the inverse polynomial method described in Appendix F.2, it is best to compare these two. First of all, Zeng et al. [31] have shown that their method is more stable than the inverse polynomial. The method meets the boundary conditions well and the thrust constraint is a big advantage. Moreover, inspecting the computation time the algorithm needs as a whole and the magnitude of the required ΔV, it is again the finite polynomial that outperforms the inverse polynomial: it takes less time to find the solution and the ΔV is closer to the optimal value.

Pro:
- It is an easy, quick and accurate method.
- It can handle a constraint on the maximum thrust acceleration.
- The number of free parameters is variable.

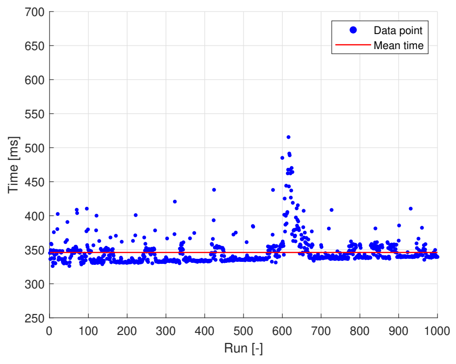
Con:
- The out-of-plane motion can only be analysed as long as the inclination does not exceed 15°.
- It can only make use of tangential thrust.

G
Uncorrected CPU-Time Measurements

This appendix contains the CPU-time measurements that underwent an outlier analysis, as their mean value is too far off from the majority of the data points. In Figure G.1 the data from the two-dimensional case I to Jupiter with a sinusoidal (Figure G.1a) and a logarithmic initialisation function (Figure G.1b) are shown. The time analysis of the two-dimensional case II to Jupiter with a tailored initialisation function where r is estimated by a power initialisation function and θ by a logarithmic initialisation function can be seen in Figure G.2.



(a) $f(\tau) = A\sin\left(\omega\left(\tau - \phi\right)\right) + K$



(b) $f(\tau) = a + b\ln(\tau)$

Figure G.1: An overview of the uncorrected average CPU-time required to find the Fourier coefficients that shape the trajectory to Jupiter.

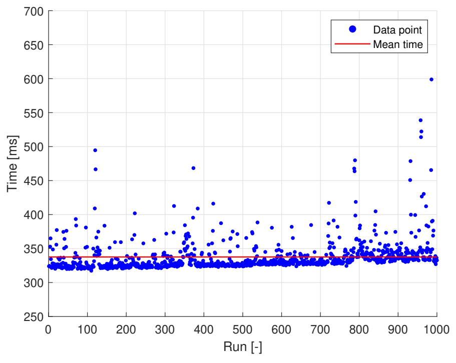


Figure G.2: The uncorrected solver convergence time of the trajectory to Jupiter that uses a polynomial function to approximate r, while a logarithmic function has been used to estimate θ.