Faculty of Physics and Astronomy
University of Heidelberg

Diploma Thesis
in Physics
submitted by

Sascha Patrick Quanz

born in Eschwege

2004

A Unified Approach to the Computation of Spacecraft Trajectories

—

A Software Tool for Mission Analysis (Simulation)

This diploma thesis has been carried out by *Sascha Patrick Quanz* at the

Astronomische Rechen-Institut
Mönchhofstrasse 12-14
69120 Heidelberg
Germany

in cooperation with the

European Space Agency (ESA)
European Space Research and Technology Center (ESTEC)
Directorate of Human Spaceflight
Development Department
Human Transportation and Re-entry Systems Division
2200 AG Noordwijk
The Netherlands

under the supervision of

Prof. Rainer Spurzem

Abstract

This diploma thesis demonstrates the feasibility of a unified physical and algorithmic model for the computation of spacecraft trajectories. It shows that one set of differential equations suffices to compute launch, re-entry, and classical satellite trajectories with high accuracy. Substantiation for this approach is provided by a functioning software implementation. Since this requires a thorough understanding of the forces acting on the spacecraft, a detailed description of these is given and the underlying physical processes are discussed. By combining atmosphere and gravitational field models with the presented unified formulation and with an appropriate numerical integrator, accurate and reliable results can be obtained for all flight phases, as is confirmed by comparison with examples from the available literature. The diploma thesis concludes with a critical review of the unified approach, including possible future applications and enhancements of the presented software tool.

Meinen Eltern

für ihre Liebe und immerwährende Unterstützung

Sascha P. Quanz

Manche Menschen sehen die Dinge, wie sie sind, und sagen:
'Warum?'
Ich träume von einigen, die es nie gab, und sage:
'Warum nicht?'
(John. F. Kennedy)

Contents

List of Figures

List of Tables

Chapter 1

Introduction

This diploma thesis deals with the computation of spacecraft trajectories. Its ambition is to expand beyond the present recognized disparity of approaches for the achievement of trajectory simulation. The current variety of methodologies and corresponding software implementations is the result of an evolution that began decades ago as briefly described in the following.

In general one can distinguish three different flight phases by physical processes that have to be taken into account: The launch phase, the orbiting phase and the atmospheric re-entry phase. Since each phase has unique physical characteristics and properties the tools for computing the trajectories have become increasingly specialized, in particular for the orbiting and re-entry phases.

The orbiting phase was mainly the field of interest for mathematicians who focused on improving the prediction accuracy of satellite trajectories with high altitudes by implementing different and sometimes very arduous and complex types of orbit perturbations. These enhanced models for the computation of so-called Keplerian orbits are used for satellites when changes in the major axes and the eccentricity of the orbit or the overall lifetime of the satellite are of major interest. Unintended changes in the orbital elements require correction maneuvers involving the firing of thrusters. In that context a detailed knowledge of the initial orbit provides the basis for an optimal maneuver. The better the correction maneuvers can be planned, the more rarely they need to be carried out, which finally leads to an increased lifetime of the satellite as the use of fuel can be minimized.

The re-entry phase was accurately analyzed by scientists and engineers, who were concerned with the safe recovery of a space vehicle. This phase requires a good knowledge of the velocity and the position of the spacecraft in order to study the deceleration and the heating during the atmospheric re-entry. Here, the classical equations of motion, as used for the computation of a Keplerian orbit, are not well suited as normally they do not consider the importance of atmospheric conditions and other physical effects like drag and lift. Consequently different approaches to describe a re-entry trajectory with high accuracy have been preferred.

In the process of improving the computation models for both phases separately, the differences between them have increased. In *[3]* in the introduction to chapter 15 *Vinh et al.* underline:

> *"The gap got wider as the two theories became more and more sophisticated. Now the two groups, one consisting mostly of mathematicians, and one consisting mostly of physicists, seldom reference the other's group work."*

This diploma thesis attempts to overcome this gap by demonstrating the suitability of a single formulation for both exo-atmospheric and re-entry computations. This still does not consider the launch phase but, as will be shown in the course of this work, also launch trajectories (for which the altitude-time profile and the related thrust-mass ratio of the launcher are of major importance) can be handled with the same and therefore "unified" physical and algorithmic formulation.

Before addressing the unified approach in detail a review of the classical description of orbits is given in the next chapter. In chapter 3 the derivation of the required set of differential equations for the unified formulation is explained in detail. The different forces acting on a spacecraft are described in chapter 4. These forces are an essential part for the computation of the trajectories. Chapter 5 deals with models of the Earth atmosphere and its gravitational field. The models are required to quantify the forces acting on the vehicle. Chapter 6 describes the general architecture of the computer program, developed in order to demonstrate the feasibility of the unified approach. Furthermore, a short introduction to the implemented numerical integration scheme is given. In chapter 7 the results of the trajectory computations are presented. Examples for all three flight phases are given and are compared to results from the available literature. Chapter 8 deals with the quality and the stability of the numerical algorithm and compares its results to those of other numerical integrators commonly used for astrophysical computations. Finally, chapter 9 summarizes the results and gives an overview of possible future enhancements and applications.

Chapter 2

The Classical Orbital Elements

For a complete solution of Kepler's equations for a two-body system (e.g. a planet and an orbiting spacecraft) six variables are required. Instead of using three cartesian variables for

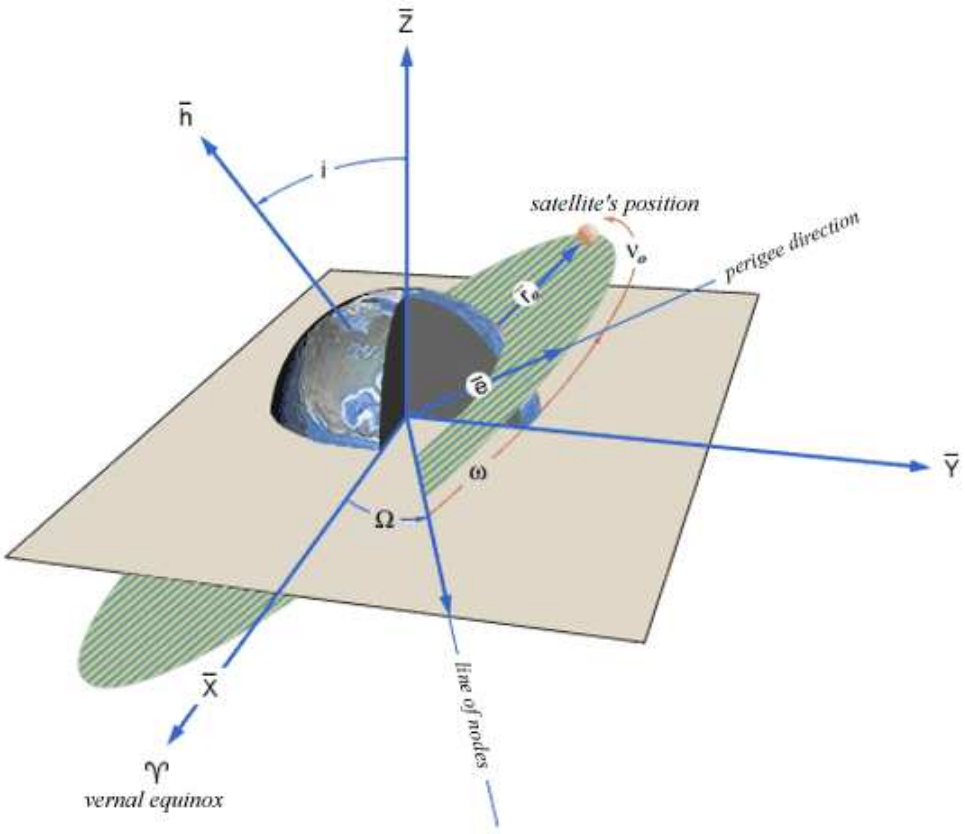

Figure 2.1: ***Definition of the classical orbital elements describing a so-called Keplerian orbit. The semi-major axis*** a ***and the eccentricity*** e ***of the orbit are not depicted. (Courtesy of ESA)***

the relative position and three for the velocity of the spacecraft, it is more convenient to use the six classical orbital elements listed in Table 2.1. They are normally used for the characterization of a pure so-called Keplerian orbit as shown in Figure 2.1.

Variable	**Meaning**
a	semi-major axis
e	eccentricity
i	inclination
Ω	right ascension of the ascending node
ω	argument of perigee
M	mean anomaly (with $M = n(t - t_0)$, n is the mean motion)

Table 2.1: ***The six classical orbital elements.***

The semi-major axis a describes the size of the ellipse while the eccentricity e describes its shape. The relation between the two parameters is given by $e = 1 - r_{peri}/a$, where r_{peri} refers to the absolute value of the radius vector when pointing in the direction of the periapsis. The inclination i is the angle between the z-axis of the equatorial plane and the angular momentum vector of the orbit. The right ascension Ω is the angle between the Vernal Equinox (the x-axis of the equatorial plane) and the ascending node, representing the point where the satellite crosses the equatorial plane coming from the south. The angle between the ascending node and r_{peri} is called the argument of perigee ω.

For non-inclined orbits ($i = 0$) the angles Ω and ω are poorly defined as the equatorial plane and the orbital plane become identical. Hence, a variation of the six orbital elements must be used. The intricacies of a detailed implementation of this variation, however, will not be addressed here.

The so-called Lagrange equations for the differential change of the six classical parameters are given below in (2.1)-(2.6). The derivation of these formulae can be found in most books on classical orbital mechanics, as for instance *[1]* and *[2]*.

$$\frac{da}{dt} = \frac{2}{na}\frac{\partial U}{\partial M} \tag{2.1}$$

$$\frac{de}{dt} = \frac{1-e^2}{na^2e}\frac{\partial U}{\partial M} - \frac{\sqrt{1-e^2}}{na^2e}\frac{\partial U}{\partial \omega} \tag{2.2}$$

$$\frac{di}{dt} = \frac{-1}{na^2\sqrt{1-e^2}\sin i}\left[\frac{\partial U}{\partial \Omega} + \cos i\frac{\partial U}{\partial \omega}\right] \tag{2.3}$$

$$\frac{d\Omega}{dt} = \frac{1}{na^2\sqrt{1-e^2}\sin i}\frac{\partial U}{\partial i} \tag{2.4}$$

$$\frac{d\omega}{dt} = \frac{\sqrt{1-e^2}}{na^2e}\frac{\partial U}{\partial e} - \frac{\cos i}{na^2\sqrt{1-e^2}\sin i}\frac{\partial U}{\partial i} \tag{2.5}$$

$$\frac{dM}{dt} = n - \frac{2}{na}\frac{\partial U}{\partial a} - \frac{1-e^2}{na^2e}\frac{\partial U}{\partial e} \tag{2.6}$$

In these equations U represents the gravitational potential whose resulting conservative force governs the satellite's trajectory. For given initial conditions the orbit of the spacecraft can be calculated by integrating the Lagrange equations with respect to time. The resulting trajectory is a perfect conic section whose shape depends only on the initial values as long

as U is spherically symmetric. Equations (2.1)-(2.6) can also be applied to the analysis of disturbed and therefore non-Keplerian orbits as long as additional disturbing forces are conservative as well (e.g. third body perturbations or differences in $\vec{F}_G$ due to non-spherical U). The influence of solar radiation pressure or atmospheric drag on the different variables, however, can not be intuitively addressed, although these effects are taken into account in most modern orbit propagation tools.

For the computation of the re-entry and the launch phase the classical orbital elements are not adequate. For these trajectories the shape of the conic section (which changes rapidly) is of minor interest and other variables like the velocity and the position of the spacecraft are more relevant. For a unified physical and mathematical approach to the computation of spacecraft trajectories a different formulation being applicable to both the exo-atmospheric and atmospheric phases must be used. Already commonly used for re-entry computations a formulation fulfilling this requirement is presented in the following chapter.

Chapter 3

Derivation of the Equations of Flight

In the following the relevant equations of motion describing a spacecraft trajectory are derived. Although currently used only for re-entry computations, these equations can be applied to all flight phases as demanded for a unified formulation.

The motion of the vehicle is defined by the position vector $\vec{r}(t)$, the velocity vector $\vec{V}(t)$ and the mass $m(t)$. At each instant, the force $\vec{F}$ acting on the spacecraft consists of three main contributions. These are the thrust $\vec{T}$, provided by the spacecraft's engines, the aerodynamic force $\vec{A}$, resulting from the presence of an atmosphere, and the gravitational force $\vec{F}_G(t, \vec{r}) = m(t) \cdot \vec{g}(\vec{r})$. The following relation results

$$\vec{F}(t, \vec{r}) = \vec{T} + \vec{A} + m(t) \cdot \vec{g}(\vec{r}) \quad . \tag{3.1}$$

Also, with respect to an inertial system, the basic vector equation applies

$$m(t) \cdot \frac{d\vec{V}}{dt} = \vec{F}(t, \vec{r}) \quad . \tag{3.2}$$

Since the spacecraft might burn fuel during in-orbit maneuvers or even eject structural components (e.g. lower stages of a rocket or the fairing), its mass is time dependent as shown in equation (3.2).

After presenting some general ideas concerning the transformation of a vector from a Galilean coordinate system to a rotating coordinate system, a suitable reference frame for computing trajectories is defined. In this reference frame the resulting vector equations based on (3.1) and (3.2) are specialized into six scalar equations that can be integrated numerically.

3.1 Relative angular motion

Consider two different coordinate systems: One is fixed and the second system is rotating with respect to the first one. Let (X-Y-Z) denote the fixed system and (I-J-K) represent the rotating system. A vector $\vec{\Lambda}$ in the rotating system can the be expressed as

$$\vec{\Lambda} = \Lambda_I \cdot \vec{e}_I + \Lambda_J \cdot \vec{e}_J + \Lambda_K \cdot \vec{e}_K \tag{3.3}$$

with the unit vectors $\vec{e}$ pointing in the direction of I, J and K. While taking the time derivative of the vector $\vec{\Lambda}$ with respect to the non-rotating system it has to be considered that the unit vectors are also time dependent. Thus,

$$\frac{d\vec{\Lambda}}{dt} = \left(\frac{d\Lambda_I}{dt}\vec{e}_I + \frac{d\Lambda_J}{dt}\vec{e}_J + \frac{d\Lambda_K}{dt}\vec{e}_K\right) + \left(\Lambda_I\frac{d\vec{e}_I}{dt} + \Lambda_J\frac{d\vec{e}_J}{dt} + \Lambda_K\frac{d\vec{e}_K}{dt}\right) \quad . \tag{3.4}$$

Since the unit vectors remain fixed with respect to the rotating system, their linear velocity can be written as

$$\frac{d\vec{e}_I}{dt} = \vec{\Omega}\times\vec{e}_I \quad , \qquad \frac{d\vec{e}_J}{dt} = \vec{\Omega}\times\vec{e}_J \quad , \qquad \frac{d\vec{e}_K}{dt} = \vec{\Omega}\times\vec{e}_K \tag{3.5}$$

where $\vec{\Omega}$ is the rotation vector. Taking this into account and defining

$$\frac{\delta\vec{\Lambda}}{\delta t} := \frac{d\Lambda_I}{dt}\vec{e}_I + \frac{d\Lambda_J}{dt}\vec{e}_J + \frac{d\Lambda_K}{dt}\vec{e}_K$$

being the time derivative of $\vec{\Lambda}$ with respect to the rotating system, equation (3.4) can be written as

$$\frac{d\vec{\Lambda}}{dt} = \frac{\delta\vec{\Lambda}}{\delta t} + \vec{\Omega}\times\vec{\Lambda} \quad . \tag{3.6}$$

Equation (3.6) describes the transformation of a vector from a stationary coordinate system to another rotating system and provides the basis for the following derivations.

3.2 The equations of motion

3.2.1 The planet centered coordinate system

Since the motion of a spacecraft with respect to the Earth is sought, the planet has to be considered as a rotating reference frame. The rotation vector $\vec{\Omega}_{IJK}$ can be defined as pointing along the K-axis with I and J describing the equatorial plane. The non-rotating coordinate system (X-Y-Z) has the same orientation. The origin of both systems is located at the center of gravity, i.e. in the center of the planet. Figure 3.1a depicts both coordinate systems in detail.

Considering the results of the previous section, the time derivative of the position vector $\vec{r}$ can be written as

$$\frac{d\vec{r}}{dt} = \vec{V} = \frac{\delta\vec{r}}{\delta t} + \vec{\Omega}\times\vec{r} \tag{3.7}$$

and for the velocity $\vec{V}$ one obtains consequently

$$\frac{d\vec{V}}{dt} = \vec{a} = \frac{\delta}{\delta t}\left(\frac{\delta\vec{r}}{\delta t} + \vec{\Omega}\times\vec{r}\right) + \vec{\Omega}\times\left(\frac{\delta\vec{r}}{\delta t} + \vec{\Omega}\times\vec{r}\right) \quad . \tag{3.8}$$

It is assumed that

$$\frac{\delta\vec{\Omega}}{\delta t} = 0 \quad .$$

Henceforth, equation (3.8) becomes

$$\frac{d\vec{V}}{dt} = \vec{a} = \frac{\delta^2\vec{r}}{\delta t^2} + 2\,\vec{\Omega}\times\frac{\delta\vec{r}}{\delta t} + \vec{\Omega}\times(\vec{\Omega}\times\vec{r}) \quad . \tag{3.9}$$

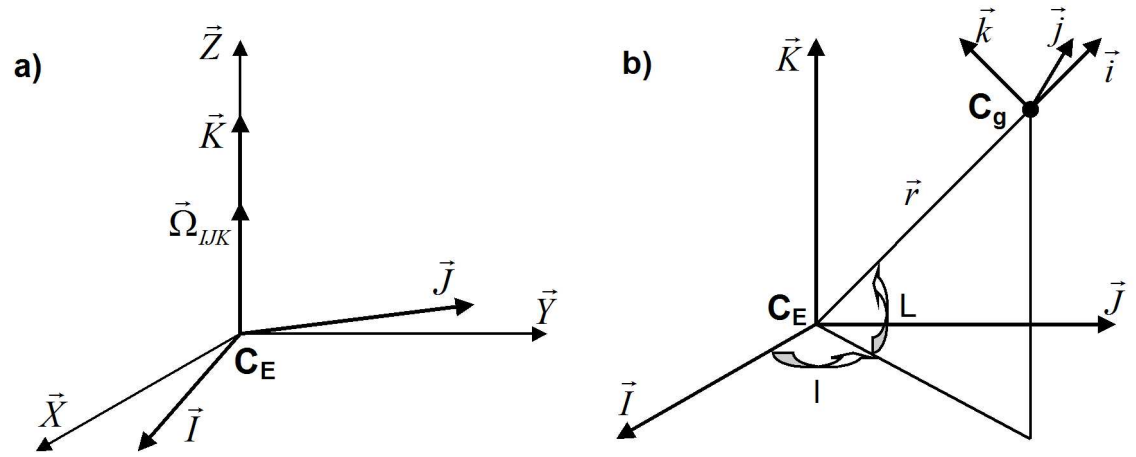

Figure 3.1: ***a) Planet centered coordinate systems: The fixed system (X-Y-Z) and the rotating system (I-J-K).*** C_E ***denotes the center of the planet (e.g. the Earth) and*** $\vec{\Omega}_{IJK}$ ***is the rotation vector pointing along the*** $\vec{K}$***- and*** $\vec{Z}$***-axis. b) Coordinate system describing the position of the spacecraft relative to the Earth.*** l ***and*** L ***are geographic longitude and geographic latitude and*** $\vec{r}$ ***is the position vector pointing from the center of the Earth to the center of gravity of the spacecraft*** C_g***, where the spacecraft centered coordinate system (i-j-k) has its origin.***

Based on (3.9) the vector equation (3.2) for the planet fixed coordinate system can be written as

$$m(t)\,\frac{d\vec{V}}{dt} = \vec{F} - 2\,m(t)\,\vec{\Omega}\times\vec{V} - m(t)\,\vec{\Omega}\times(\vec{\Omega}\times\vec{r}) \quad , \tag{3.10}$$

where $\vec{V}$ is the velocity relative to the Earth's rotating atmosphere and the time derivative is taken with respect to the Earth fixed coordinate system.

3.2.2 The spacecraft centered coordinate system

In order to evaluate the different vectors in equation (3.10) a new coordinate system (i-j-k) linked to the spacecraft is defined. The origin of this coordinate system is located at the center of gravity of the spacecraft C_g. The i-axis is pointing along the position vector $\vec{r}$, the j-axis lies in the equatorial plane (I-J) and is positive in the direction of motion and the k-axis completes the right-handed system. The basis vectors can be written as

$$\vec{i} = \frac{\overrightarrow{C_E C_g}}{r} \quad , \qquad \vec{k} = \frac{\partial\vec{i}}{\partial L} \quad , \qquad \vec{j} = \vec{k}\times\vec{i} \quad .$$

Figure 3.1b shows the position of the spacecraft relative to the Earth and also the connected coordinate system (i-j-k). C_E stands for the center of the Earth and (I-J-K) is the equatorial reference frame. L and l can be identified as geographic latitude and geographic longitude.

3.2.3 The kinematic equations

Having introduced suitable coordinate systems the first three equations governing the motion of a spacecraft can be derived. These are the so-called kinematic equations. Figure 3.1b

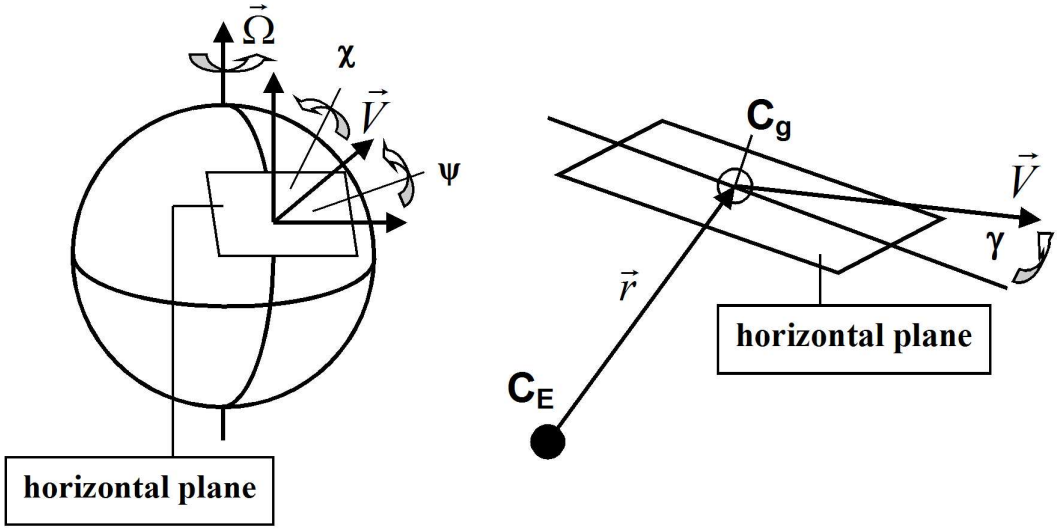

Figure 3.2: ***Definition of the flight path angle*** γ***, the heading*** ψ ***and the azimuth of the relative velocity*** χ***. The center of gravity of the spacecraft*** C_g ***lies in the horizontal plane defined to be perpendicular to the radius vector*** $\vec{r}$***.***

shows that the position of the spacecraft relative to the Earth centered coordinate system is defined by the norm of the position vector $\vec{r}$, the latitude L, and the longitude l. However, in order to describe the orientation of the velocity vector $\vec{V}$ relative to the spacecraft centered coordinate system (and thus relative to the planet centered system as well), three additional variables have to be introduced. First, the flight path angle γ measured between the local horizontal plane (that is, the plane passing through the vehicle and being orthogonal to the position vector $\vec{r}$) and the velocity vector $\vec{V}$. Second, the heading ψ being the angle between the local parallel of latitude and the projection of $\vec{V}$ on the horizontal plane. And finally, the azimuth of the relative velocity χ defined as the angle between the direction of local north and the projection of $\vec{V}$ on the local horizontal plane. From these definitions it arises that the sum of azimuth χ and heading ψ always results in 90°. Figure 3.2 shows the three angles in detail.

Referring to the spacecraft centered coordinate system, one can now write

$$\vec{r} = r\,\vec{e}_i \tag{3.11}$$

and respectively

$$\vec{V} = (V\,\sin\gamma)\,\vec{e}_i + (V\,\cos\gamma\,\sin\chi)\,\vec{e}_j + (V\,\cos\gamma\,\cos\chi)\,\vec{e}_k \quad . \tag{3.12}$$

In the same reference frame the angular velocity of the Earth can be written as

$$\vec{\Omega} = (\Omega\,\sin L)\,\vec{e}_i + (\Omega\,\cos L)\,\vec{e}_k \quad . \tag{3.13}$$

Thus, one gets

$$\begin{aligned}\vec{\Omega}\times\vec{V} \;=\; & -(\Omega\,V\,\cos\gamma\cos L\sin\chi)\,\vec{e}_i + \Omega\,V\,(\sin\gamma\cos L - \cos\gamma\sin L\cos\chi)\,\vec{e}_j \\ & +(\Omega\,V\,\cos\gamma\sin L\sin\chi)\,\vec{e}_k\end{aligned} \tag{3.14}$$

and accordingly

$$\vec{\Omega} \times (\vec{\Omega} \times \vec{r}) = -(\Omega^2 \, r \, \cos^2 L) \, \vec{e}_i + (\Omega^2 \, r \sin L \cos L) \, \vec{e}_k \quad . \tag{3.15}$$

In order to take the time derivative of the vectors $\vec{r}$ and $\vec{V}$ with respect to the planet fixed system (I-J-K) the angular velocity vector $\vec{\Omega}'$ of the spacecraft centered coordinate system (i-j-k) needs to be evaluated. The spacecraft centered system can be obtained from the fixed system by a rotation l about the positive K-axis followed by a rotation L about the negative J-axis. Hence, the angular velocity $\vec{\Omega}'$ of the rotating system (i-j-k) is

$$\vec{\Omega}' = \left(\sin L \, \frac{dl}{dt} \right) \vec{e}_i - \left(\frac{dL}{dt} \right) \vec{e}_j + \left(\cos L \, \frac{dl}{dt} \right) \vec{e}_k \quad . \tag{3.16}$$

Following equation (3.5) the time derivatives of the unit vectors of the spacecraft centered coordinate system are obtained

$$\frac{d\vec{e}_i}{dt} = \vec{\Omega}' \times \vec{e}_i = \left(\cos L \, \frac{dl}{dt} \right) \vec{e}_j + \left(\frac{dL}{dt} \right) \vec{e}_k \tag{3.17}$$

$$\frac{d\vec{e}_j}{dt} = \vec{\Omega}' \times \vec{e}_j = -\left(\cos L \, \frac{dl}{dt} \right) \vec{e}_i + \left(\sin L \, \frac{dl}{dt} \right) \vec{e}_k \tag{3.18}$$

$$\frac{d\vec{e}_k}{dt} = \vec{\Omega}' \times \vec{e}_k = -\left(\frac{dL}{dt} \right) \vec{e}_i - \left(\sin L \, \frac{dl}{dt} \right) \vec{e}_j \quad . \tag{3.19}$$

Taking into account the results in (3.17)-(3.19) the time derivative of the position vector $\vec{r}$ can now be expressed via

$$\frac{d\vec{r}}{dt} = \vec{V} = \left(\frac{dr}{dt} \right) \vec{e}_i + \left(r \, \cos L \, \frac{dl}{dt} \right) \vec{e}_j + \left(r \, \frac{dL}{dt} \right) \vec{e}_k \quad . \tag{3.20}$$

Comparing (3.20) to (3.12) leads directly to three scalar equations, the so called kinematic equations:

$$\frac{dr}{dt} = V \, \sin \gamma \tag{3.21}$$

$$\frac{dl}{dt} = \frac{V \, \cos \gamma \sin \chi}{r \, \cos L} \tag{3.22}$$

$$\frac{dL}{dt} = \frac{V \, \cos \gamma \cos \chi}{r} \tag{3.23}$$

3.2.4 The force equations

Three out of the six equations needed to describe the path of a body moving through space have been found in (3.21)-(3.23). Three more equations are required and they will be derived in this section. For that purpose use is made of equation (3.1) in connection with a description of the forces acting on a spacecraft in the spacecraft centered coordinate system.

For the gravitational force one gets directly

$$\vec{F}_G(t, \vec{r}) = -m(t) \, g(\vec{r}) \, \vec{e}_i \quad . \tag{3.24}$$

In the following it is assumed that the aerodynamic force $\vec{A}$ and the thrust $\vec{T}$ can be projected into one component acting parallel to the velocity vector $\vec{V}$ and another component acting perpendicular to it. The latter one lies in the ($\vec{r}$-$\vec{V}$)-plane. Hence, a tangential force $\vec{F}_T$ and a normal force $\vec{F}_N$, acting along and in a direction orthogonal to $\vec{V}$ respectively, can be defined. A detailed qualitative analysis of these forces is given in the next chapter.

Following equation (3.12) the tangential force can be written as

$$\vec{F}_T = (F_T \sin\gamma)\,\vec{e}_i + (F_T \cos\gamma \sin\chi)\,\vec{e}_j + (F_T \cos\gamma \cos\chi)\,\vec{e}_k \quad . \tag{3.25}$$

If the trajectory is contained in one single plane (i.e. no changes in the orbital plane during the flight), the vector $\vec{F}_N$ lies always in the ($\vec{r}$-$\vec{V}$)-plane. In this case a lateral component of the force does not exist. If, however, the vector $\vec{F}_N$ is rotated about the velocity vector $\vec{V}$, a lateral component of $\vec{F}_N$ is created. This leads to an out-of-plane motion. The resulting angle between the vertical plane, i.e. the ($\vec{r}$-$\vec{V}$)-plane, and the normal force $\vec{F}_N$ is called the bank angle μ. Now, in order to identify the components of the force $\vec{F}_N$ with respect to the spacecraft centered coordinate system (i-j-k), a primed coordinate system (i'-j'-k') is introduced. If a bank angle $\mu \neq 0$ is considered, the normal force can be split in a component $F_N \cos(\mu)$ lying in the vertical plane and being orthogonal to $\vec{V}$ and a second component $F_N \sin(\mu)$ being orthogonal to the vertical plane. Based on these components the primed coordinate system is defined by i' pointing in the direction of $F_N \cos(\mu)$, j' being parallel to $\vec{V}$ and k' pointing along $F_N \sin(\mu)$. Figure 3.3 shows the primed coordinate system, as well as the connected forces and the bank angle. Formally the primed system (i'-j'-k') can be deduced from the system (i-j-k) by a rotation ψ in the horizontal plane, followed by a rotation γ in the vertical plane. Both rotations can be described by the following transformation matrix equation

$$\begin{pmatrix} i \\ j \\ k \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos\psi & -\sin\psi \\ 0 & \sin\psi & \cos\psi \end{pmatrix} \begin{pmatrix} \cos\gamma & \sin\gamma & 0 \\ -\sin\gamma & \cos\gamma & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} i' \\ j' \\ k' \end{pmatrix}$$

or

$$\begin{pmatrix} i \\ j \\ k \end{pmatrix} = \begin{pmatrix} \cos\gamma & \sin\gamma & 0 \\ -\sin\gamma\cos\psi & \cos\gamma\cos\psi & -\sin\psi \\ -\sin\gamma\sin\psi & \cos\gamma\sin\psi & \cos\psi \end{pmatrix} \begin{pmatrix} i' \\ j' \\ k' \end{pmatrix} \quad .$$

Since in the primed system the components of the normal force $\vec{F}_N$ are

$$i' = F_N \cos\mu \quad , \qquad j' = 0 \quad , \qquad k' = F_N \sin\mu \tag{3.26}$$

this force can be written with respect to the spacecraft centered coordinate system as

$$\begin{aligned} \vec{F}_N \;=\; & (F_N \cos\mu\cos\gamma)\,\vec{e}_i - (F_N \cos\mu\sin\gamma\cos\psi + F_N \sin\mu\sin\psi)\,\vec{e}_j \\ & - (F_N \cos\mu\sin\gamma\sin\psi - F_N \sin\mu\cos\psi)\,\vec{e}_k \end{aligned} \tag{3.27}$$

or equivalently

$$\begin{aligned} \vec{F}_N \;=\; & (F_N \cos\mu\cos\gamma)\,\vec{e}_i - (F_N \cos\mu\sin\gamma\sin\chi + F_N \sin\mu\cos\chi)\,\vec{e}_j \\ & - (F_N \cos\mu\sin\gamma\cos\chi - F_N \sin\mu\sin\chi)\,\vec{e}_k \end{aligned} \tag{3.28}$$

if the azimuth of the relative velocity χ is considered instead of the heading ψ. Keeping this in mind, equation (3.12) can be revisited and the time derivative of the velocity vector $\vec{V}$ can

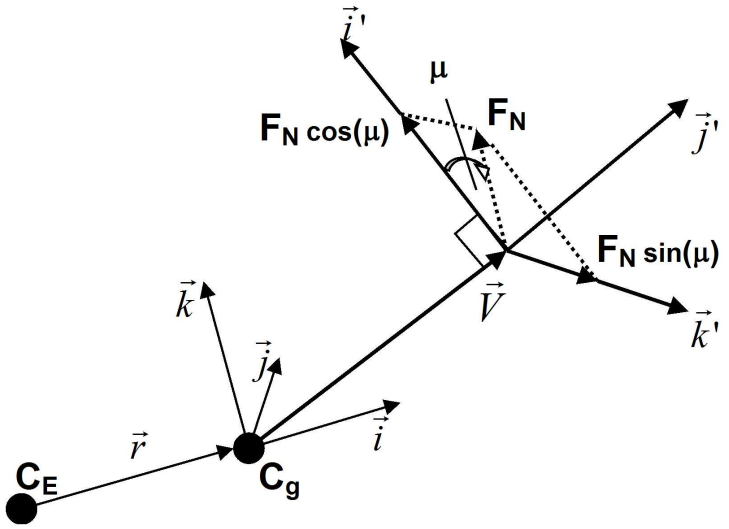

Figure 3.3: ***Definition of the primed coordinate system linked to the velocity vector. If the normal force $\vec{F}_N$, usually lying in the vertical plane and being perpendicular to the velocity, is rotated about the velocity vector, the resulting angle between the normal force and the vertical plane is called the bank angle μ. The newly created lateral component of the force $F_N \sin(\mu)$ leads to an out-of-plane motion.***

be taken. To that end the relations given in (3.17)-(3.19), (3.22) and (3.23) are used resulting in

$$\begin{aligned}
\frac{d\vec{V}}{dt} &= \left(\sin\gamma \frac{dV}{dt} + V\,\cos\gamma \frac{d\gamma}{dt} - \frac{V^2}{r}\cos^2\gamma \right) \vec{e}_i \\
&+ \Bigg(\cos\gamma \sin\chi \frac{dV}{dt} - V\,\sin\gamma \sin\chi \frac{d\gamma}{dt} - V\,\cos\gamma \cos\chi \frac{d\chi}{dt} \\
&+ \frac{V^2}{r}\cos\gamma \sin\chi \Big(\sin\gamma - \cos\gamma \cos\chi \tan L \Big) \Bigg) \vec{e}_j \\
&+ \Bigg(\cos\gamma \cos\chi \frac{dV}{dt} - V\,\sin\gamma \cos\chi \frac{d\gamma}{dt} + V\,\cos\gamma \sin\chi \frac{d\chi}{dt} \\
&+ \frac{V^2}{r}\cos\gamma \Big(\sin\gamma \cos\chi + \cos\gamma \sin^2\chi \tan L \Big) \Bigg) \vec{e}_k \quad .
\end{aligned} \tag{3.29}$$

By substituting into the basic equation (3.10) and considering equations (3.14) and (3.15) as well as those for the forces (3.24), (3.25) and (3.28), one obtains the following three scalar equations:

$$\begin{aligned}
&\sin\gamma \frac{dV}{dt} + V\,\cos\gamma \frac{d\gamma}{dt} - \frac{V^2}{r}\cos^2\gamma \\
&= \frac{F_T}{m}\sin\gamma + \frac{F_N}{m}\cos\mu\cos\gamma - g + 2\,\Omega\,V\,\cos\gamma\cos L\sin\chi + \Omega^2\,r\,\cos^2 L
\end{aligned} \tag{3.30}$$

$$\begin{aligned}
&\cos\gamma\frac{dV}{dt} - V\,\sin\gamma\frac{d\gamma}{dt} - V\,\cos\gamma\cot\chi\frac{d\chi}{dt} + \frac{V^2}{r}\cos\gamma\Big(\sin\gamma - \cos\gamma\cos\chi\tan L\Big)\\
&= \frac{F_T}{m}\cos\gamma - \frac{1}{m}\Big(F_N\,\cos\mu\sin\gamma + F_N\,\sin\mu\cot\chi\Big)\\
&\quad -\frac{2\,\Omega\,V}{\sin\chi}\Big(\sin\gamma\cos L - \cos\gamma\sin L\cos\chi\Big)
\end{aligned} \tag{3.31}$$

$$\begin{aligned}
&\cos\gamma\frac{dV}{dt} - V\,\sin\gamma\frac{d\gamma}{dt} + \frac{V\,\cos\gamma}{\cot\chi}\frac{d\chi}{dt} + \frac{V^2}{r}\cos\gamma\left(\sin\gamma + \frac{\cos\gamma\sin\chi\tan L}{\cot\chi}\right)\\
&= \frac{F_T}{m}\cos\gamma - \frac{1}{m}\left(F_N\,\cos\mu\sin\gamma - \frac{F_N\,\sin\mu}{\cot\chi}\right)\\
&\quad -2\,\Omega\,V\frac{\cos\gamma\sin L}{\cot\chi} - \Omega^2\,r\frac{\sin L\cos L}{\cos\chi}
\end{aligned} \tag{3.32}$$

These equations can be solved for the derivatives dV/dt, $d\gamma/dt$ and $d\chi/dt$ leading to

$$\frac{dV}{dt} = -g\sin\gamma + \frac{1}{m}F_T + \Omega^2\,r\cos L\Big(\sin\gamma\cos L - \cos\gamma\sin L\cos\chi\Big) \tag{3.33}$$

$$\begin{aligned}
\frac{d\gamma}{dt} &= -\frac{g}{V}\cos\gamma + \frac{1}{m\,V}F_N\cos\mu + \frac{V}{r}\cos\gamma + 2\,\Omega\cos L\sin\chi\\
&\quad +\frac{\Omega^2\,r\cos L}{V}\Big(\cos\gamma\cos L + \sin\gamma\sin L\cos\chi\Big)
\end{aligned} \tag{3.34}$$

$$\begin{aligned}
\frac{d\chi}{dt} &= -\frac{1}{m\,V}F_N\frac{\sin\mu}{\cos\gamma} + \frac{V}{r}\cos\gamma\tan L\sin\chi\\
&\quad +2\,\Omega\Big(\sin L - \tan\gamma\cos L\cos\chi\Big) + \frac{\Omega^2\,r}{V}\frac{\sin L\cos L\sin\chi}{\cos\gamma}
\end{aligned} \tag{3.35}$$

These three equations are normally referred to as force equations and together with (3.21)-(3.23) they form a set of six differential equations describing the motion of a spacecraft with respect to a rotating planet, e.g. the Earth.

The derived equations cover all types of trajectories relevant for the course of this work. But before this can be demonstrated a better understanding of the different forces a vehicle encounters on its way through space is required. Therefore the next chapter describes the origin, influence and contribution to F_N and F_T of these forces.

Chapter 4

Forces Acting on a Spacecraft

Prior to a description of the different forces and their influence on a spacecraft, a detailed look at the spacecraft itself is provided by Figure 4.1 representing a generic spacecraft configuration (not necessarily representative of a launch vehicle!). x and z define a coordinate system having

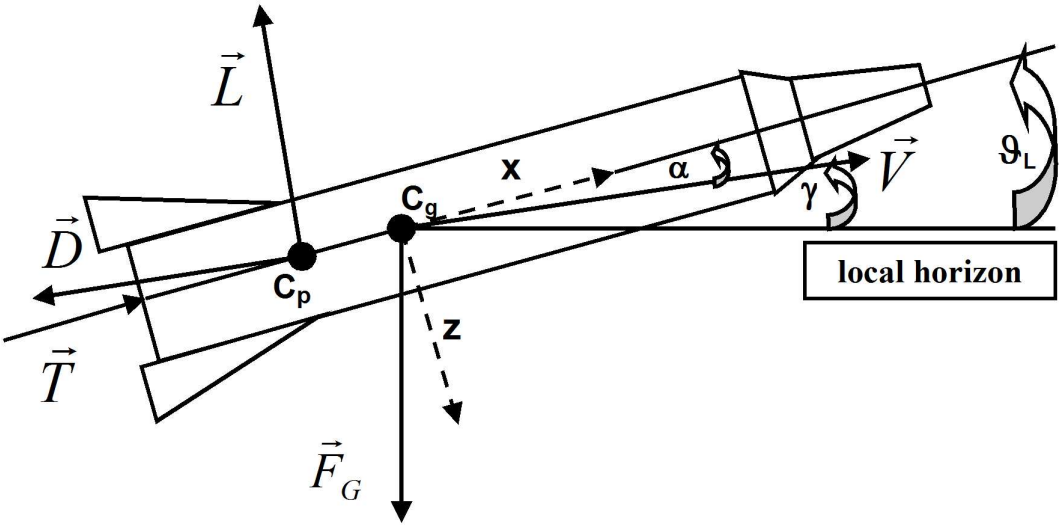

Figure 4.1: ***Typical spacecraft configuration and related forces.*** *x and z define a coordinate system related to the spacecraft's symmetry axis. The thrust $\vec{T}$ is normally aligned with the main body axis x. C_p indicates the center of pressure where the aerodynamic force is applied. This force can be split into lift $\vec{L}$ and drag $\vec{D}$ components. C_g represents the center of gravity of the spacecraft and $\vec{F}_G$ the gravitational force. α is the angle of attack, measured between the x-axis and the velocity $\vec{V}$. γ represents the flight path angle, already introduced in the previous chapter. ϑ_L is the local pitch angle and is defined between the local horizon and the main body axis of the spacecraft.*

its origin in the center of gravity of the spacecraft C_g with x running along the main body axis and z lying in the vertical plane. C_p indicates the center of pressure, the point at which the resulting aerodynamic force acts. The flight path angle γ is the angle between the local

horizon and the velocity vector $\vec{V}$. The angle of attack α is measured between the velocity vector and the x-axis of the spacecraft. By positioning the center of gravity ahead the center of pressure the spacecraft is stabilized as for increasing α the aerodynamic force creates a restoring moment about the center of gravity that tends to bring the spacecraft back to its initial attitude. The local pitch angle ϑ_L is defined as the angle between the local horizon and the x-axis. This angle plays a major role in the computation of launch trajectories as by following a preprogrammed pitch law, as part of the guidance strategy, the rocket is led to the desired injection parameters (see also chapter 6.1.1,5 and chapter 7.3).

Keeping these definitions in mind the different forces acting on the spacecraft can be introduced. These forces have numerous origins and they can vary over a wide range of values by orders of magnitude depending on the actual flight regime. The main contributions to a resulting force are:

- The gravitational force $\vec{F}_G$, acting at the center of gravity. For orbits with high altitudes or long orbit durations not only the gravitational pull of the Earth but also the gravitational influence of the moon, the sun and other planets do effect the trajectory.
- The aerodynamic force $\vec{A}$, acting at the center of pressure. It can be decomposed into a lift $\vec{L}$, a drag $\vec{D}$ and a side force component. The latter one is not depicted in Figure 4.1. The aerodynamic force results from the presence of an atmosphere and has great influence during the ascent and the atmospheric re-entry of the vehicle where its magnitude is comparable to that of the gravitational force.
- The thrust $\vec{T}$, delivered by the spacecraft's engines. This vector is usually aligned with the x-axis of the spacecraft for structural reasons, although its direction can vary due to gimballing of the engines' nozzles.

For high precision orbit and trajectory predictions also other contributions like solar and Earth radiation pressure or relativistic effects have to be taken into account. A detailed analysis of these and other additional forces can be found in *[14]*. The main focus here is on the three forces mentioned above. However, only the gravitational influence of the Earth is considered is the following while that of the sun and the moon is neglected. That this it at least justified for low Earth orbits (LEO) is shown in Figure 4.2, where a comparison between the influence of different forces is provided.

Following the assumptions made above a resulting force can be written as

$$\vec{F}_{result} = \vec{F}_G + \vec{A} + \vec{T} \tag{4.1}$$

as previously defined in (3.1). A representation of the aerodynamic force $\vec{A}$ in terms of its tangential and normal components $\vec{F_T}$ and $\vec{F_N}$ is detailed in the following pages.

4.1 The aerodynamic force $\vec{A}$

4.1.1 Drag, lift and side force and related moments about a reference reduction center

Essentially, the aerodynamic force results from friction and pressure forces. With respect to the (x-y-z) coordinate system defined in Figure 4.1 the aerodynamic force can be split into a so-called axial force (for which another nomination is tangential force) $\vec{F}_A$, pointing in

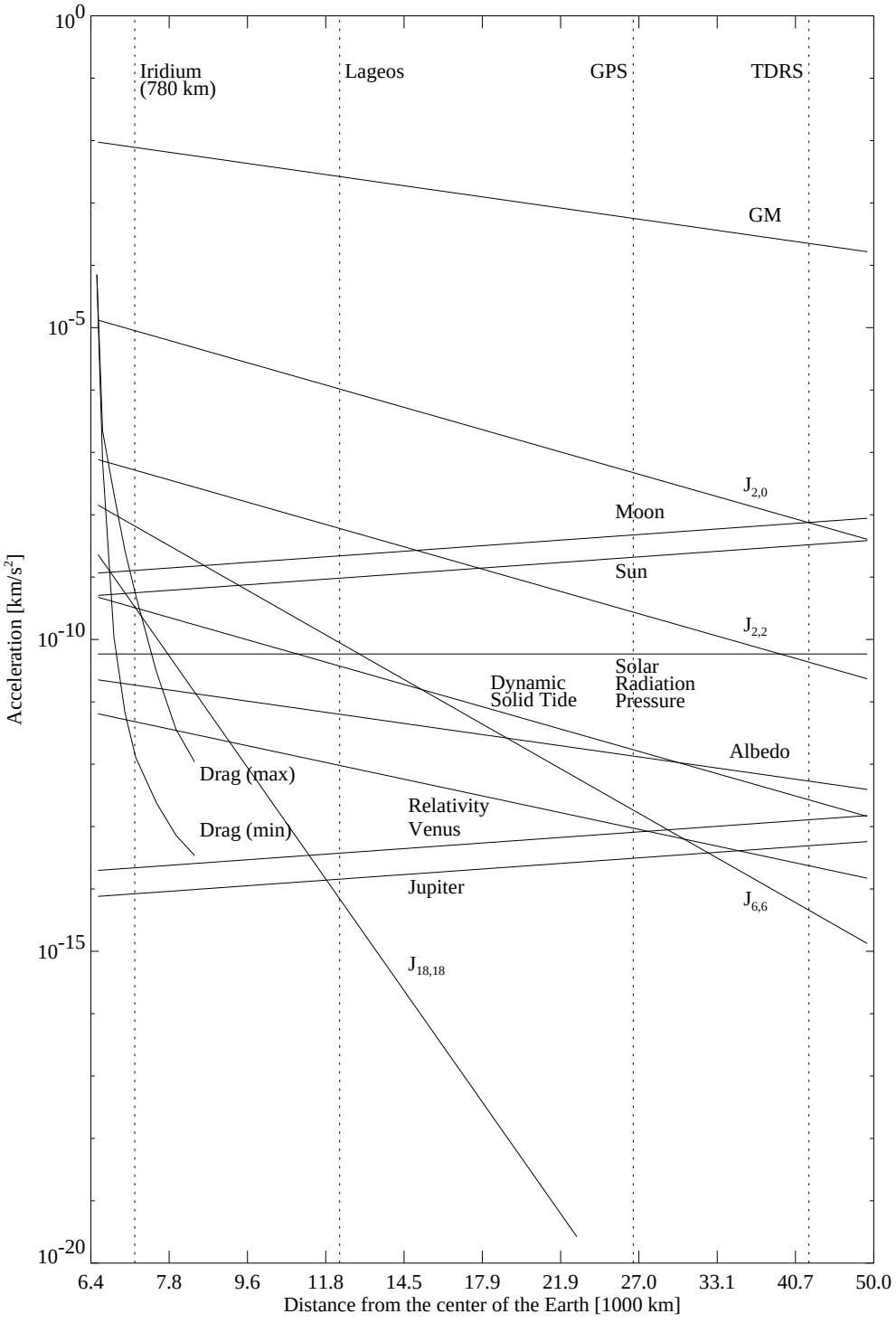

Figure 4.2: ***Comparison of different forces acting on a spacecraft.*** *GM* ***refers to the Earth gravitational field with*** $J_{2,0}$, $J_{2,2}$, $J_{6,6}$ ***and*** $J_{18,18}$ ***being correction terms due to the oblateness of the Earth. Furthermore the strength of the gravitational pull of other celestial bodies, the influence of the Earth albedo and the magnitude of relativistic effects are shown. For LEOs the influence of the moon, the sun and other celestial bodies can be neglected. (Taken from [8] with corrections from the author)***

negative x-direction, a normal force $\vec{F}_N$, pointing in negative z-direction and a side force $\vec{F}_S$, pointing along the y-axis. Mathematically these forces are defined as

$$\vec{F}_A = -\frac{1}{2}\rho S C_A V^2 \vec{e}_x \quad , \qquad \vec{F}_N = -\frac{1}{2}\rho S C_N V^2 \vec{e}_z \quad , \qquad \vec{F}_S = \frac{1}{2}\rho S C_S V^2 \vec{e}_y \tag{4.2}$$

with S being a reference area, ρ standing for the upstream air density, V being the norm of the relative velocity vector and C being the aerodynamic coefficient for each force. More information about the aerodynamic coefficients is given in the next section.

In the same reference frame one can define different moments related to the axes: The rolling-moment, around the x-axis of the spacecraft, the pitching-moment around the y-axis and the yawing-moment around the z-axis. Figure 4.3 shows the reference frame and the related moments and forces.

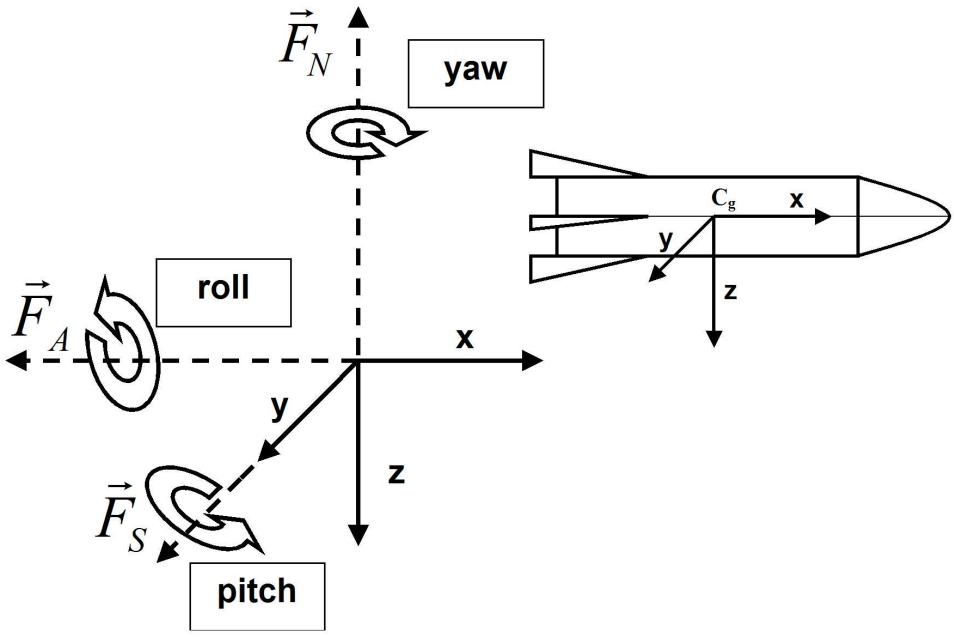

Figure 4.3: ***Definition of the different components of the aerodynamic force for a coordinate system linked to the spacecraft main body axis. For each axis the related moment is depicted.***

While the different moments are always defined abound the spacecraft body axes, a different coordinate system for the decomposition of the aerodynamic force can be specified. One possible reference frame is linked to the velocity. Considering the force equations (3.33)-(3.35) this reference frame proves to be most suitable and it leads directly to the general definition of lift and drag (see also Figure 4.1).

Drag is defined to be the component of the aerodynamic force opposing the direction of motion. Therefore it is antiparallel to the velocity vector. Lift is the component of the force that is perpendicular to the velocity and also perpendicular to the y-axis. The third component, not represented in Figure 4.1 (it would point outwards of the page), is still called the side force. Following the definitions in (4.2) drag and lift can be written as

$$\vec{D} = -\frac{1}{2}\rho S C_D V^2 \vec{e}_V \quad , \qquad \vec{L} = \frac{1}{2}\rho S C_L V^2 \vec{e}_L \tag{4.3}$$

with $\vec{e}_V$ pointing in the direction of the velocity and $\vec{e}_L$ being the unit vector in the direction of the lift. Again, S is a reference area, ρ is the upstream air density and V represents the norm of the relative velocity vector.

4.1.2 The aerodynamic coefficients

From (4.3) the drag coefficient C_D and the lift coefficient C_L result in

$$C_D = \frac{2D}{S\rho V^2} \quad , \qquad C_L = \frac{2L}{S\rho V^2} \quad . \tag{4.4}$$

These coefficients refer to the total aerodynamic force like those of equation (4.2), but generally they assume different values due to the different reference frames. It should be emphasized that the computation of the aerodynamic coefficients, e.g. C_L and C_D, for a real spacecraft can be challenging. In particular for a re-entry vehicle different flight regimes featuring different environmental conditions must be considered as the spacecraft goes from hypersonic (with speeds roughly in excess of Mach 6) and supersonic speed through transonic and subsonic speed for the approach and landing. In order to compute the global aerodynamic coefficients the actual force acting on every surface element of the spacecraft must be calculated, before it is possible to integrate over the whole area. Here, the friction and the pressure of the streaming air do not only depend on the altitude and attitude of the spacecraft, as they also change with the position on the spacecraft, where they are measured. Consequently, a general formulation of the aerodynamic coefficients does usually depend on the angle of attack, the Mach number and also on the Reynolds number of the upstream flow.

4.1.3 The computation of aerodynamic properties

In order to compute the aerodynamic properties of a spacecraft the governing equations of fluid mechanics are required. They are derived from statements of the conservation of mass, momentum and energy for an arbitrary control volume. For cylindrical coordinates the following Navier-Stokes equations in conservation forms are obtained:

$$\frac{\partial}{\partial t}(W) + \nabla F(W) = \nabla N(W) + H(W) \tag{4.5}$$

with

$$W = \begin{pmatrix} \rho \\ \rho u_z \\ \rho u_r \\ \rho E \end{pmatrix}$$

being the vector of the conservation variables ρ (density), u_r and u_z (radial and axial component of the velocity) and E (total energy). The other terms in (4.5) can be expressed as

$$F_z(W) = \begin{pmatrix} \rho u_z \\ \rho u_z^2 + p \\ \rho u_z u_r \\ (\rho E + p) u_z \end{pmatrix}, \quad F_r(W) = \begin{pmatrix} \rho u_r \\ \rho u_r u_z \\ \rho u_r^2 + p \\ (\rho E + p) u_r \end{pmatrix}, \quad N_z(W) = \begin{pmatrix} 0 \\ \tau_{zz} \\ \tau_{zr} \\ u_z \tau_{zz} + u_r \tau_{zr} \end{pmatrix},$$

$$N_r(W) = \begin{pmatrix} 0 \\ \tau_{rz} \\ \tau_{rr} \\ u_z\tau_{zr} + u_r\tau_{rr} \end{pmatrix} \quad \text{and} \quad H(W) = \begin{pmatrix} 0 \\ 0 \\ -\frac{\tau_{\theta\theta}}{r} \\ 0 \end{pmatrix}$$

with p representing the pressure and the included tensor components

$$\tau_{zz} = \mu_{tot}\left[2\frac{\partial u_z}{\partial z} - \frac{2}{3}(\nabla u)\right], \qquad \tau_{zr} = \mu_{tot}\left[\frac{\partial u_z}{\partial r} + \frac{\partial u_r}{\partial z}\right],$$

$$\tau_{rr} = \mu_{tot}\left[2\frac{\partial u_r}{\partial r} - \frac{2}{3}(\nabla u)\right], \quad \tau_{\theta\theta} = \mu_{tot}\left[2\frac{u_r}{r} - \frac{2}{3}(\nabla u)\right] \quad .$$

Here, μ is the dynamic viscosity of the fluid. These equations are based on the assumption that the flow is axis symmetric with respect to the spacecraft body axis. Turbulence is not considered. The treatment of turbulence is usually accomplished by taking into account additional transport equations of variables relevant for the definition of an effective turbulent viscosity, e.g. k and ϵ which result in an equivalent (increased) flow viscosity. In *[25]* a complete derivation of the Navier-Stokes equations can be found.

Figure 4.4 and Figure 4.5 show the results of a typical computation of aerodynamic properties for a generic spacecraft: Figure 4.4 represents the distribution of the pressure coefficient, while Figure 4.5 depicts the connected friction lines and heat fluxes. These figures were generated with a laminar Navier-Stokes approach, assuming perfect gas conditions for the hypersonic flight regime (Mach 6). The angle of attack was $\alpha = 40°$. By varying these conditions different values and distributions for the pressure coefficient and the friction lines are obtained. From these results the aerodynamic coefficients can be derived as a function of the Mach number and the angle of attack. In order to illustrate the complexity of the task Table 4.1 provides a comparison of different aerodynamic prediction methods.

Level	**Type**	**Limitation**	**Complexity**	**Computation Time**
0	empirical	qualitative	algebraic	seconds
1	linear	small perturbations	algebraic	minutes
2	inviscid	no separation	differentials	hour(s)
3	Navier-Stokes	no restriction	partial differential	hours/days

Table 4.1: ***Properties of aerodynamic prediction methods. (Taken from [25])***

4.2 The thrust $\vec{T}$

As previously mentioned the main thrust of the spacecraft is normally aligned with the main body axis for structural reasons. However, it is possible to change the direction of the thrust to a certain amount by gimballing the engines' nozzles, assisting in the execution of flight maneuvers.

In a reference frame linked to the velocity vector, the tangential and normal components of the thrust can be expressed as

$$\vec{T}_T = T\cos(\alpha)\,\vec{e}_V \quad , \quad \vec{T}_N = T\sin(\alpha)\,\vec{e}_L \tag{4.6}$$

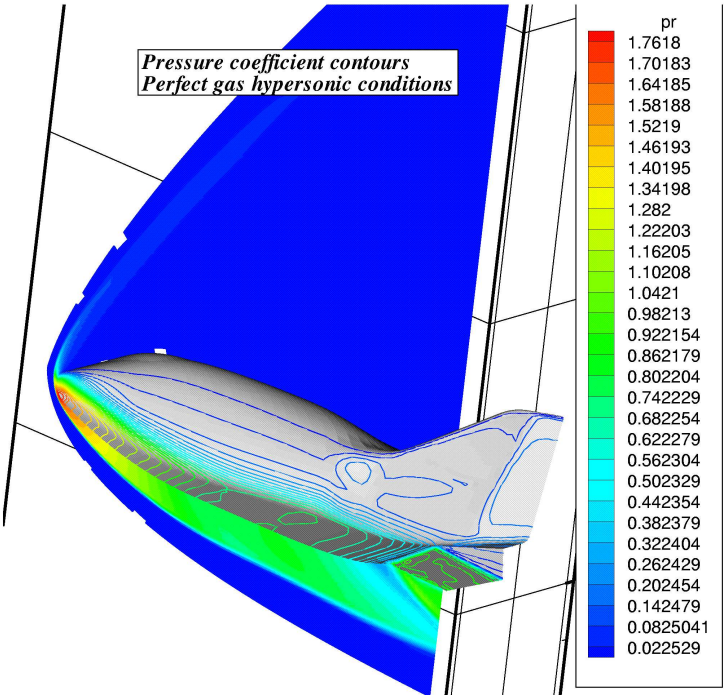

Figure 4.4: ***Pressure coefficient distribution for a generic re-entry vehicle for the hypersonic flight regime with an angle of attack of*** 40°. ***(Courtesy of ESA)***

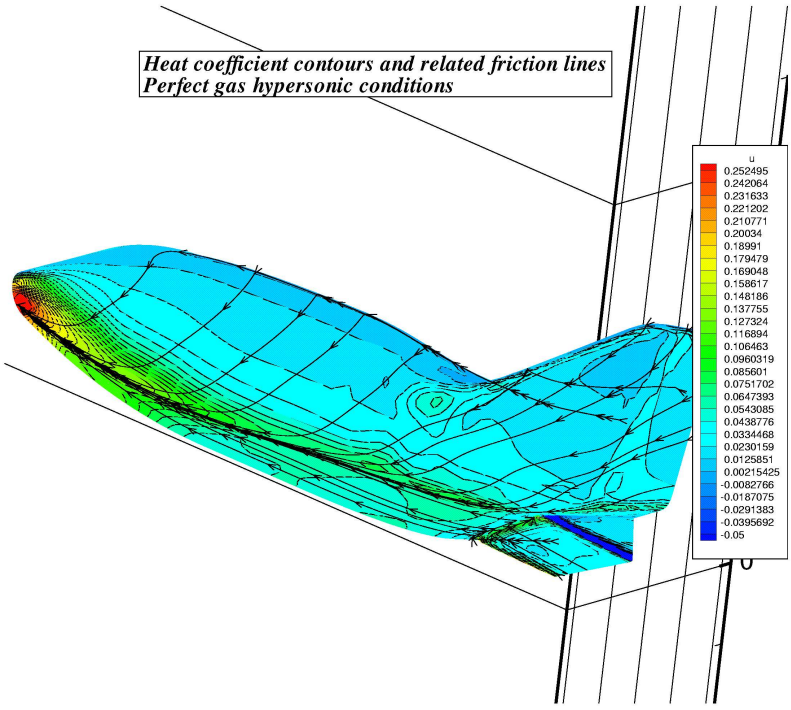

Figure 4.5: ***Friction lines and heat fluxes for a generic re-entry vehicle for the hypersonic flight regime with an angle of attack of*** 40°. ***(Courtesy of ESA)***

with $\vec{T}_T$ being aligned with the velocity vector and $\vec{T}_N$ being perpendicular to it. This equation can be rewritten as

$$\vec{T}_T = T\cos(\vartheta_L - \gamma)\,\vec{e}_V \quad , \quad \vec{T}_N = T\sin(\vartheta_L - \gamma)\,\vec{e}_L \tag{4.7}$$

by using the local pitch angle ϑ_L and the flight path angle γ instead of the angle of attack α (see Figure 4.1). The formulation in (4.7) is of particular relevance for the computation of launch trajectories, as during the ascent the pitch angle is imposed. The flight path angle then results from this preprogrammed pitch law summarizing the guidance principles for the fulfillment of the required injection parameters.

4.3 The resulting tangential and normal forces $\vec{F}_T$ and $\vec{F}_N$

Taking into account the results of the previous sections the force equations (3.33)-(3.35) can be revisited. Since the tangential force $\vec{F}_T$ and the normal force $\vec{F}_N$ are defined with respect to the velocity vector, it can be written

$$\vec{F}_T = T\cos(\alpha)\,\vec{e}_V - \vec{D} \quad , \quad \vec{F}_N = T\sin(\alpha)\,\vec{e}_L + \vec{L} \quad . \tag{4.8}$$

Or, more precisely

$$\vec{F}_T = \Big(T\cos(\alpha) - \frac{1}{2}\rho S C_D V^2\Big)\vec{e}_V \tag{4.9}$$

and

$$\vec{F}_N = \Big(T\sin(\alpha) + \frac{1}{2}\rho S C_L V^2\Big)\vec{e}_L \quad . \tag{4.10}$$

In accordance with (4.7) these equations can be written for the computation of a launch trajectory as

$$\vec{F}_T = \Big(T\cos(\vartheta_L - \gamma) - \frac{1}{2}\rho S C_D V^2\Big)\vec{e}_V \tag{4.11}$$

and

$$\vec{F}_N = \Big(T\sin(\vartheta_L - \gamma) + \frac{1}{2}\rho S C_L V^2\Big)\vec{e}_L \quad . \tag{4.12}$$

From these equations it is apparent, that no other forces than gravity act on the spacecraft as soon as the thrust is nullified and the influence of an atmosphere is neglected ($\rho = 0$). Consequently, in that case an unperturbed Keplerian orbit should be obtained provided that proper initial conditions are given.

Before showing examples of such Keplerian trajectories confirming the validity of this assumption (chapter 7), the following pages discuss the gravitational field of the Earth and also the properties of the planet's atmosphere. Accurate physical models of both are required as they directly influence the spacecraft trajectory as seen in this chapter.

Chapter 5

Atmosphere and Gravitational Field of the Earth

The gravitational potential of celestial bodies is of major relevance for a spacecraft whose trajectory is in their vicinity. However, if the celestial body possesses an atmosphere and if the operations of the spacecraft involve significant portions in that region, atmospheric effects can not be neglected either. Consequently, a brief introduction to models of the Earth's atmosphere and a description of the geoid's gravitational field is provided in this chapter.

5.1 Models of the Earth atmosphere

Different analytical and empirical atmosphere models are commonly used for the computation of spacecraft trajectories and orbits. They provide the required information about air density and air temperature. As seen in (4.3) the air density contributes directly to the aerodynamic force $\vec{A}$, and hence also to its components $\vec{L}$ and $\vec{D}$. The air temperature is needed to compute the speed of sound leading to the actual Mach number of the spacecraft. Since the aerodynamic coefficients depend on the Mach number (see 4.1.2) accessing its value is mandatory.

The speed of sound can be calculated via

$$v_s = \sqrt{\frac{\kappa\, c\, \Theta}{m_{mol}}} \tag{5.1}$$

with $\kappa = c_P/c_V$ being the ratio of the specific heat coefficients, c representing the specific gas constant, Θ being the absolute temperature and $m_{mol} = 28.96\, g$ representing the mass of one mole of air. The Mach number follows via

$$M = \frac{V}{v_s} \quad , \tag{5.2}$$

where V is the velocity of the spacecraft relative to the atmosphere. Thus, V corresponds directly to the velocity used in the set of differential equations (3.21)-(3.23) and (3.33)-(3.35).

Within this section the focus is now on three different atmosphere models, being used for the computation of the examples shown in chapter 7.

5.1.1 No atmosphere

For the calculation of perfect and unperturbed Keplerian orbits the atmospheric influence is neglected. Therefore, the atmospheric density is set to $\rho = 0$ for all altitudes.

5.1.2 An analytical model

The following expressions for the atmospheric density provide an appropriate analytical approximation. Here, r is the altitude above the Earth surface in kilometer and the air density ρ is given in kg/m^3:

$$\begin{array}{lll} r > 155: & \rho = 3.5 \cdot 10^{-12} \cdot \exp(-(r-380)/48) & \text{(see also } [10]) \\ r \leq 155: & \rho = \rho_0 \cdot \exp(-900 \cdot r/R_{ea}) & \text{(see also } [12]) \end{array}$$

In the last equation $\rho_0 = 1.225\, kg/m^3$ stands for the air density at ground level and R_{ea} is the mean Earth radius. Figure 5.1 shows the resulting plot for altitudes up to 400 km.

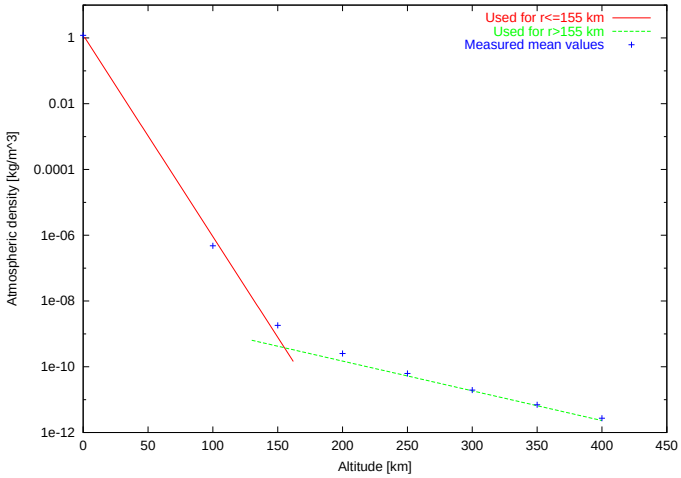

Figure 5.1: ***Analytical model for the atmospheric density. The measured mean values are taken from [7].***

Concerning the temperature profile of the Earth atmosphere it is more difficult to find an analytical expression. However, the following piecewise linear approximations can be used for an estimation of the absolute air temperature as a function of the altitude r.

$$\begin{array}{lcl} 0 < r \leq 11 & : & \Theta = -6.3636 \cdot r + 287 \\ 11 < r \leq 15 & : & \Theta = -0.25 \cdot r + 219.75 \\ 15 < r \leq 25 & : & \Theta = 0.1 \cdot r + 214.5 \\ 25 < r \leq 48 & : & \Theta = 2.8261 \cdot r + 146.35 \\ 48 < r \leq 50 & : & \Theta = 0.5 \cdot r + 258 \\ 50 < r \leq 53 & : & \Theta = -1/3 \cdot r + 299.67 \\ 53 < r \leq 80 & : & \Theta = -4.2963 \cdot r + 509.7 \\ 80 < r \leq 85 & : & \Theta = -1.2 \cdot r + 262 \\ 85 < r \leq 97 & : & \Theta = 0.5 \cdot +117.5 \\ 97 < r \leq 170 & : & \Theta = 8.6849 \cdot r - 676.44 \\ 170 < r \leq 220 & : & \Theta = 2.8 \cdot r + 324 \\ 220 < r \leq 500 & : & \Theta = 0.2143 \cdot r + 892.86 \end{array}$$

Again, the altitude above the Earth surface is measured in kilometer and the temperature Θ is given in Kelvin.

In section 5.2 a comparison between the analytical expressions provided above and the results of an empirical atmosphere model described in the following section will demonstrate that the analytical descriptions serve as a good approximation.

5.1.3 The empirical NRLMSISE-00 model

This empirical atmosphere model from the Naval Research Laboratory in Washington DC developed by *J.M. Picone et al.* is based on the earlier MSISE90 model and is one of the most advanced models for the Earth atmosphere (see *[6]*). Since in this model the density and the temperature of the atmosphere depend on the time, on the position and also on solar radiation fluxes, the model can be profitably used for high accuracy long term orbit predictions. However, the model is by far more powerful than it is required as only the values of the total mass density and the absolute temperature are of direct interest. Possible features like Oxygen number density, Hydrogen number density or exospheric temperature (but to name a few) are of no immediate relevance for the computation of the trajectory. In order to compute the desired values of ρ and Θ some input information is required. The input of the original NRLMSISE-00 model consists of the nine variables shown in Table 5.1.

Variable	**Meaning [Unit]**
DOY	Day of the Year
UT	Universal Time [*sec*]
ALT	Altitude [*km*]
GLAT	Latitude [*deg*]
GLONG	Longitude [*deg*]
STL	Local apparent solar time [*sec*]
AF107	The 81-day average of the 10.7 *cm* solar flux centered on DOY
F107	The 10.7 *cm* solar flux of the previous day
AP	The daily magnetic index

Table 5.1: ***Original input variables for the NRLMSISE-00 atmosphere model. In the developed software program the actual year was added as an input variable in order to assure correct access to two solar data bases describing fluctuations in the solar activity.***

The position variables, i.e. the latitude, the longitude and the altitude, are inherent in the system of differential equations. The date and the actual time have to be specified additionally. The local apparent solar time can easily be calculated by using

$$STL = UT/3600 + GLONG/15 \quad . \tag{5.3}$$

The last three input variables in Table 5.1 describe the solar activity and resulting fluctuations of the geomagnetic field. Although the 10.7 *cm* solar flux does not contribute directly to a deformation of the Earth's upper atmosphere or ionosphere (it is not energetic enough), it is an important indicator of the solar activity as it tends to follow the changes in the solar ultraviolet. That region of the solar spectrum influences the composition and the density of the Earth atmosphere immensely. The magnetic index A_p describes the daily fluctuations

of the geomagnetic field ranging from 0 nT (quiet) to 400 nT (greatly disturbed). These perturbations, also caused by solar radiation, can be used as an indicator for solar activity as well. NRLMSISE-00 uses these parameters as input in order to determine deformations of the atmosphere from which it derives the related air density.

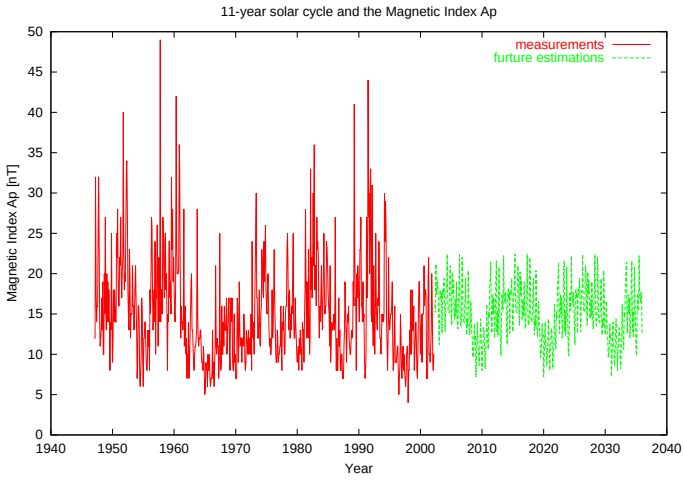

Figure 5.2: ***Values of the magnetic index A_p as measured in the past (since 1947) and extrapolated estimations for the future.***

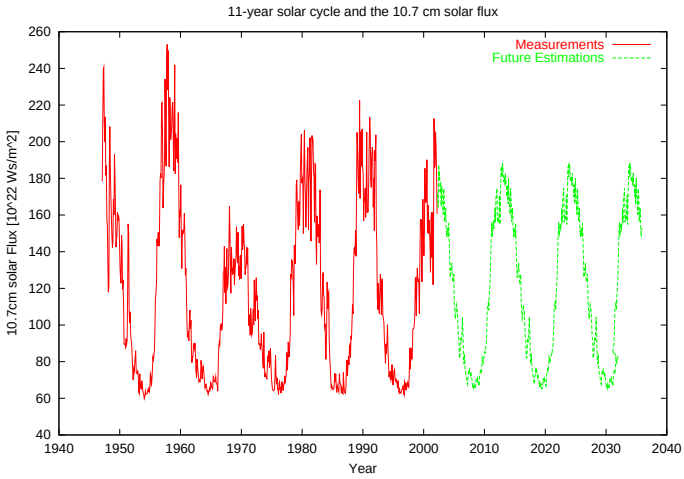

Figure 5.3: ***Values of the 10.7 cm solar flux as measured in the past (since 1947) and extrapolated estimations for the future.***

The software tool used for the computation of the trajectories in chapter 7 includes two databases that have been added to the original NRLMSISE-00 model in order to obtain an enhanced and more realistic model of the atmosphere. The databases contain daily values for the 10.7 cm solar flux and for the magnetic index A_p from 1960 until March 2002. They are freely available on the Internet[1] and can be updated easily if required. By adding these two

[1] ftp://ftp.ngdc.noaa.gov/STP/SOLAR_DATA/SOLAR_RADIO/FLUX/
ftp://ftp.ngdc.noaa.gov/STP/GEOMAGNETIC_DATA/INDICES/KP_AP/

databases, all input variables of the model, shown in Table 5.1, can be quantified, as the values for A_p and for the 10.7 *cm* flux can be read from the databases directly and the 81-day average of the solar flux can be calculated easily. However, the databases cover only the time period until March 2002. As a result, trajectories generated for any time later than March 2002 can not be treated so far. This problem can be solved by using the monthly average of the solar flux and the magnetic index from 1960 until 2002. By averaging over corresponding months it is possible to reproduce the 11-year solar cycle that can be extrapolated into the future. Thus, by relying on earlier measured values contained in the databases, an enhanced NRLMSISE-00 atmosphere model can predict the values of the 10.7 *cm* flux and the magnetic index for any time in the future. However, it should be emphasized that these are only estimations as it is impossible to predict the solar activity accurately. Figures 5.2 and 5.3 show the variations of the solar flux, those of the magnetic index and also the resulting estimated values for the next decades. Since the enhanced atmosphere model should be able to compute the total atmospheric density and the temperature for any given time, the actual year has to be added as an input variable to the original NRLMSISE-00 model (see Table 5.1). This has been done in the developed software tool so that the model can now compute the values of ρ and Θ depending on the 10.7 *cm* flux and the magnetic index A_p for any given time.

Some additional background information as well as a link to a FORTRAN source code of the original NRLMSISE-00 model can be found on the Internet[2]. The source code in C based on the FORTRAN version can also be downloaded from the Internet[3]. However, one should note that the C version of the code was not officially published by the Naval Research Laboratory.

5.2 Comparison of the atmosphere models

Figure 5.4 and Figure 5.5 show clearly the influence of the solar activity on the atmospheric properties, when they are computed with the NRLMSISE-00 atmosphere model. For the

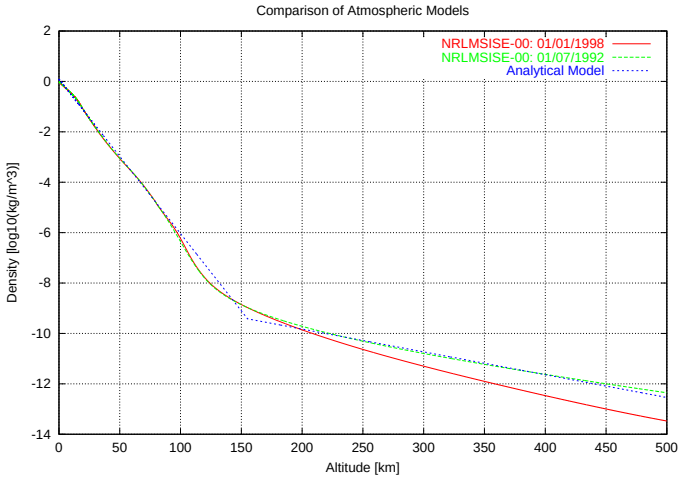

Figure 5.4: ***Density profile of the Earth atmosphere for altitudes from ground level to 500 km as calculated by the analytical model and by NRLMSISE-00. For the latter one the results for two different years are shown.***

[2]http://nssdc.gsfc.nasa.gov/space/model/atmos/nrlmsise00.html
[3]http://www.brodo.de/english/pub/nrlmsise/

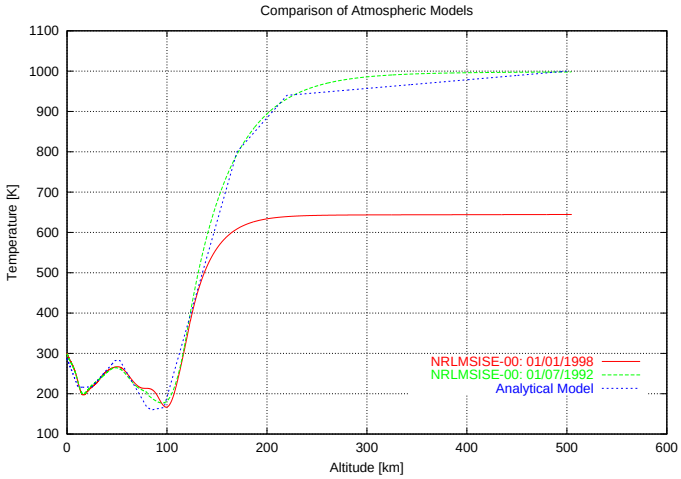

Figure 5.5: ***Temperature profile of the Earth atmosphere for altitudes from ground level to 500 km as calculated by the analytical model and by NRLMSISE-00. For the latter one the results for two different years are shown.***

comparison a year with high solar activity (1992) and a year of low activity (1998) as well as different months were chosen. The values of the analytical atmosphere models are shown in parallel.

For low altitudes no big differences are found neither in the temperature nor in the density profile of all three computations. For high altitudes, however, the influence of the solar activity is clearly visible. Especially the density increases by a factor of 10 in times of high solar activity when computed with NRLMSISE-00. In consequence, this increase influences directly the components of the aerodynamic force as can be seen in (4.3).

In addition it becomes obvious that the analytical models for the density and the temperature tend to follow the results of the empirical model for times of high solar activity. In that respect the analytical models describe a rather dense and hot atmosphere. This should be kept in mind if they are used for trajectory computations.

5.3 The gravitational field of the Earth

Since the Earth is not a perfect sphere and since its mass is distributed inhomogeneously, the gravitational potential is not spherically symmetric either. This effect is often referred to as oblateness of the Earth. By using a spherical harmonic expansion a convenient description of the gravitational field at a distance r is (see also *[7]*):

$$\begin{aligned} U(r,l,L) \;=\; & \frac{\mu_{ea}}{r}\Bigg\{-1+\sum_{n=2}^{\infty}\Bigg[\left(\frac{R_{ea}}{r}\right)^{n} J_n P_{n0}\Big(\cos(L)\Big) \\ & +\sum_{m=1}^{n}\left(\frac{R_{ea}}{r}\right)^{n}\Big(C_{nm}\cos(ml)+S_{nm}\sin(ml)\Big)P_{nm}\Big(\cos(L)\Big)\Bigg]\Bigg\} \end{aligned} \tag{5.4}$$

The gravitational parameter μ_{ea} of the Earth can be written as

$$\mu_{ea} = m_{ea}\cdot\gamma_G \tag{5.5}$$

with m_{ea} being the mass of the Earth and γ_G representing the gravitational constant. L and l are geographic latitude and longitude and P_{nm} Legendre polynomials. J_n, S_{nm} and C_{nm} are the so-called zonal, tesseral and sectoral coefficients of the Earth and are based on its mass distribution. In Table 5.2 the magnitude of low order J, S and C are given. In many cases the

Zonal	**Tesseral**	**Sectoral**
$J_2 = 1082.6 \cdot 10^{-6}$	$C_{21} = 0$	$S_{21} = 0$
$J_3 = -2.53 \cdot 10^{-6}$	$C_{22} = 1.57 \cdot 10^{-6}$	$S_{22} = -0.904 \cdot 10^{-6}$
$J_4 = -1.62 \cdot 10^{-6}$	$C_{31} = 2.19 \cdot 10^{-6}$	$S_{31} = 0.27 \cdot 10^{-6}$
$J_5 = -0.23 \cdot 10^{-6}$	$C_{32} = 0.31 \cdot 10^{-6}$	$S_{32} = -0.21 \cdot 10^{-6}$
$J_6 = 0.54 \cdot 10^{-6}$	$C_{33} = 0.1 \cdot 10^{-6}$	$S_{33} = 0.197 \cdot 10^{-6}$

Table 5.2: ***Zonal harmonic, tesseral harmonic and sectoral harmonic coefficients of the Earth. (Taken from [7])***

first or second order approximations are sufficient, as the J_2-term dominates by far the other terms (see Figure 4.2). Especially for launch and re-entry computations high order correction terms can be neglected. For high accuracy long term orbit predictions, however, it might be necessary to consider also higher order terms. In *[14]* a list containing the coefficients up to the 20^{th} order can be found.

With the discussed models of the atmosphere and the gravitational field of the Earth all required information for the computation of spacecraft trajectories has been presented. How this information can be combined and implemented in a software tool is shown in the following chapter.

Chapter 6

The Implementation

This chapter deals with the architecture of the computer program that is used for the computation of the spacecraft trajectories presented in chapter 7. It describes in detail the input and output files as well as the Runge-Kutta algorithm, a numerical integrator scheme being the core part of the program. The complete source code of the program in C can be found in the appendices.

6.1 The general program architecture

A program to compute trajectories can be implemented in many different ways. This section, however, describes one possible program architecture based on the unified approach combining comfortable handling of input and output data with an easily understandable logical structure. A flowchart showing the program structure in detail is presented in Figure 6.2 later in this chapter. Since in this flowchart also the different input and output files are depicted a brief description of each file is given in the following.

6.1.1 The input files

In order to make the program more flexible, the input consists of different files. This includes files for

- the specification of the trajectory phase (e.g. orbit, launch or re-entry),
- the characterization of the spacecraft (e.g. mass, surface, aerodynamic coefficients),
- the specification of the physical model (e.g. atmosphere and gravitation model),
- the definition of the stop condition for the numerical loop (e.g. maximum propagation time).

The name, the content and the purpose of each input file is described hereafter. In the appendices an example for each file is given.

1) launch_input.txt, orbit_input.txt, manual_input.txt

These input files define which kind of trajectory is to be computed. One of these files has to be selected at the beginning of the program.

a: launch_input.txt: This file contains all the information required for the computation of a launch trajectory. In Table 6.1 the different input variables and parameters are shown. They include the characteristics of the rocket (e.g. thrust, burn duration of the stages) as well as the initial values for the latitude, the longitude and for the azimuth of the relative velocity. This launch azimuth is important as it is directly connected to the inclination of the desired orbit. Since the radius vector $\vec{r}$ is measured from the center of the Earth, the initial altitude r_0 is ground level, i.e. the Earth radius. In order to avoid a division by zero in (3.34) and (3.35), the initial relative velocity V_0 has to be set to a very small non-zero value. Both is done by the program automatically. Thus, the initial altitude and the initial relative velocity are already specified without being input parameters. The initial flight path angle $\gamma_0 = 90°$ is also set automatically. The evolution of the flight path angle γ during the launch results from the imposed pitch law (see chapter 4.2 and 6.1.1.5).

Variable/Parameter	**Meaning [Unit]**
General Information	
L_0	Initial/Launch pad latitude $[deg]$
l_0	Initial/Launch pad longitude $[deg]$
χ_0	Initial/Launch azimuth of the relative velocity $[deg]$
t_{lo}	Lift-off time $[s]$ (after ignition)
n_{stages}	Number of stages
$m_{payload}$	Payload mass $[kg]$
$N_{addmass}$	Additional Mass Flag:
	Indicates how many additional mass
	items are following (see below)
For each stage:	
$n_{engines}$	Number of engines
T_e	Average thrust per engines $[N]$
β	Angle between thrust and rocket main axis $[deg]$
I_s	Average specific impulse $[m/s]$
t_{start}	Burn start time $[s]$ (after ignition)
t_{stop}	Burn stop time $[s]$ (after ignition)
m_{prop}	Propellant mass of stage $[kg]$
t_{sep}	Separation time $[s]$ (after ignition)
m_{sep}	Separation mass $[kg]$
For each additional mass item:	
m_{add}	Additional mass $[kg]$
t^a_{start}	Beginning of mass separation $[s]$ (after ignition)
t^a_{stop}	End of mass separation $[s]$ (after ignition)

Table 6.1: ***Input variables and parameters of launch_input.txt***

Concerning the input for the rocket boosters one should note that the thrust T_e and the specific impulse I_s of an engine usually depend on the atmospheric pressure and therefore on the altitude. However, in this input file average values of these parameters should be provided

as a correct modelling of the engines' performance is a rather cumbersome task.

Another important aspect of a launch trajectory is the evolution of the mass during the flight. The mass of the rocket consists of the payload mass, the actual propellant mass, the mass of the structure and possible additional mass items like water for the cooling of the nozzles. For the propellant mass it is assumed that the mass flow rate is constant. Then it can be calculated by

$$\frac{dm}{dt} = \frac{T}{I_s} \quad . \tag{6.1}$$

The following examples illustrate the purpose of the additional mass items: The jettisoning of a fairing can be simulated by putting t^a_{start} equal to t^a_{stop} while m_{add} is representing the mass of the fairing. The loss of cooling water can be simulated by defining the specific cooling period via t^a_{start} and t^a_{stop} and by putting m_{add} equal to the mass of the water lost during that period. For the computation of the actual mass the mass flow rate is assumed to be constant.

b: orbit_input.txt: This input file is selected if a classical satellite orbit or a combined orbit-and-re-entry trajectory is to be computed. The input variables and parameters are summarized in Table 6.2. The computation is started either at minimum or maximum lat-

Variable/Parameter	**Meaning [Unit]**
r_{apo}	Apogee above Earth surface $[km]$
r_{peri}	Perigee above Earth surface $[km]$
l_0	Initial longitude $[deg]$
i	Inclination of the orbit $[deg]$
N_{lati}	Latitude Flag: Computation can be started at 1) Perigee and minimum latitude 2) Perigee and maximum latitude 3) Apogee and minimum latitude 4) Apogee and maximum latitude
N_{direc}	Direction Flag: Computation can be started 1) Going eastwards 2) Going westwards
$N_{deorbit}$	De-orbit Flag: Indicates whether a de-orbit maneuver shall be initiated or not (see 5.1.1.2)

Table 6.2: ***Input variables and parameters of orbit_input.txt***

itude in the perigee or apogee. Hence, the initial flight path angle is always $\gamma_0 = 0°$. The initial relative velocity V_0 is well defined by the input parameters as it can be derived from the absolute velocities at apogee or perigee for which one finds

$$V^{apo}_{abs} = \sqrt{\frac{\mu_{ea}}{r_{apo}}}\sqrt{\frac{2r_{peri}}{r_{peri}+r_{apo}}} \quad , \tag{6.2}$$

$$V^{peri}_{abs} = \sqrt{\frac{\mu_{ea}}{r_{peri}}}\sqrt{\frac{2r_{apo}}{r_{peri}+r_{apo}}} \tag{6.3}$$

with the gravitational parameter

$$\mu_{ea} = \gamma_G \cdot m_{ea} \approx 3.98941 \cdot 10^{14} \frac{m^3}{s^2} \quad . \tag{6.4}$$

Here, γ_G is the gravitational constant and m_{ea} represents the mass of the Earth. The norm of the initial relative velocity can then be computed via

$$V_0 = V_{abs} \pm \Omega \cdot r_0 \cdot L_0 \tag{6.5}$$

with $\Omega \approx 7.2921159 \cdot 10^{-5}\ rad/s$ being the angular velocity of the Earth and r_0 and L_0 representing the initial altitude and the initial latitude. The rotation of the Earth has to be either added or subtracted from the absolute velocity depending on the direction of the spacecraft. If the trajectory leads eastwards the rotation has to be subtracted, for a westward flight it has to be added. The flight direction, defined by the Direction Flag N_{direc}, drives also the initial value of the azimuth of the relative velocity. For a flight in eastward direction the azimuth is $\chi_0 = 90°$, while for a westward flight it is $\chi_0 = 270°$. The inclination i defines the absolute value of the initial latitude L_0, while the Latitude Flag N_{lati} defines its sign and also the initial value of the altitude r_0 (see Table 6.2). Finally, the De-orbit Flag $N_{deorbit}$ indicates whether a de-orbit maneuver shall be carried out in order to leave the orbit and to re-enter the atmosphere at a foreseen location. For the execution of the de-orbit maneuver, however, additional information is needed. This information is stored in a different input file called *Deorbit.txt*. It is described in detail in 6.1.1.2 later in this chapter.

c: manual_input.txt: This input file offers the possibility to define all initial values manually. The input variables and parameters are shown in Table 6.3. Like in the previous input

Variable/Parameter	**Meaning [Unit]**
r_0	Initial altitude $[km]$
V_0	Initial relative velocity $[m/s]$
γ_0	Initial flight path angle $[deg]$
χ_0	Initial azimuth of the relative velocity $[deg]$
L_0	Initial latitude $[deg]$
l_0	Initial longitude $[deg]$
$N_{deorbit}$	De-orbit Flag:
	Indicates whether a de-orbit
	maneuver is planned or not

Table 6.3: ***Input variables and parameters of manual_input.txt***

file, the De-orbit Flag $N_{deorbit}$ specifies whether a de-orbit maneuver is supposed to be initiated or not. The required information for the computation of this maneuver is stored in the input file following hereafter.

2) Deorbit.txt

This input file is needed for the computation of a simple de-orbit maneuver leading the spacecraft from its initial orbit to a defined landing site. In order to be able to touch down

at the right location on the ground, the spacecraft aims already for a so-called entry interface being a specific point within the outer regions of the atmosphere. Passing through this entry interface assures that the final landing site on the ground can be reached. Thus, if one finds a way to get from the initial orbit to the right entry interface, the correct landing spot can hardly be missed. An easy solution for this problem is described in the following.

Parameter	**Meaning [Unit]**
l_{deo}	Entry interface longitude $[deg]$
L_{deo}	Entry interface latitude $[deg]$
r_{deo}	Entry interface altitude $[km]$
γ_{deo}	Entry interface flight path angle $[deg]$

Table 6.4: ***Input parameters of Deorbit.txt***

The entry interface is defined by the parameters contained in this input file and shown in Table 6.4. In addition to the three spatial coordinates (altitude, latitude and longitude), the desired flight path angle at that point has also to be specified. For simplification it is assumed that the entry interface (and also the final landing spot) lies in the plane of the initial orbit. The concrete values for the parameters of the entry interface depend on the foreseen landing site and also on spacecraft properties. They can either be determined via an iterating process (e.g. the atmospheric arc from the entry interface to the ground is computed until the exact coordinates are found), or they are already known and can be specified directly.

In order reach the so-defined entry interface, the spacecraft has to leave its initial orbit by initiating a so-called de-orbit burn. This de-orbit burn consists of a specific decrease of the relative velocity (a negative ΔV). If the time period for firing the thrusters, required to decelerate the vehicle, is short compared to the time needed to complete one further orbit, the computation of the ΔV is based on the impulsive maneuver hypothesis and the reduction of the velocity can be assumed to take place instantaneously.

Based the values for r_{deo} and γ_{deo} (see Table 6.4) and knowing the current values of r, V and γ it is possible to compute a ΔV that leads the spacecraft from its initial position to the desired values of r_{deo} and γ_{deo} by using the following second degree equation

$$\begin{aligned}
&\left[\frac{r_0^2}{r_{deo}^2\cos(\gamma_{deo})}\Big(\sin^2(\omega+\gamma_0)-1\Big)+1\right](\Delta V)^2 \\
&+2V_0\left[\cos\omega-\frac{r_0^2}{r_{deo}^2\cos(\gamma_{deo})}\cos\gamma_0\cos(\omega+\gamma_{deo})\right]\Delta V \\
&+2\mu_{ea}\left[\frac{1}{r_{deo}}-\frac{1}{r_0}\right]+V_0^2\left[1-\frac{r_0^2}{r_{deo}^2\cos^2(\gamma_{deo})}\cos^2\gamma_0\right]=0
\end{aligned} \tag{6.6}$$

Here, the variables with the index 0 represent the initial values, i.e. before the maneuver, and ω is the angle between the vector of the initial velocity and ΔV. Since it was assumed that the maneuver takes place within the initial orbital plane, one finds $\omega = 180$. As one can see in equation (6.6), the ΔV does not depend on the latitude or the longitude, neither of the initial orbit nor of the entry interface. However, in order to make sure that the de-orbit burn $V_0 - \Delta V = V'$ leads to a successful and correct re-entry, a test computation is carried

out: If after the maneuver the correct values for l_{deo} and L_{deo} are reached, the de-orbit burn can also be initiated in the main program. If the values are not reached, the de-orbit burn is postponed and the spacecraft continues to fly its initial orbit. Figure 6.1 illustrates the purpose of the test computation.

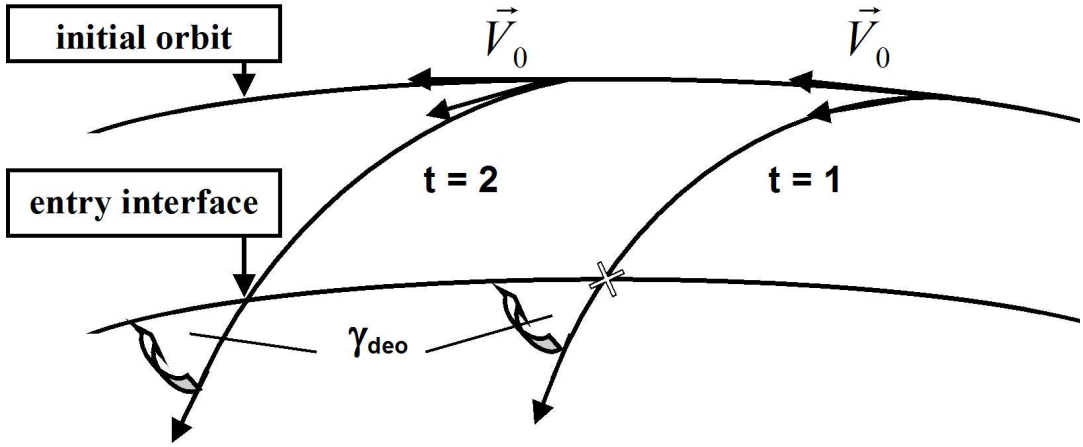

Figure 6.1: ***Illustration of the connection between initial orbit, de-orbit maneuver and entry interface. It is assumed that the entry interface lies in the same plane as the initial orbit. Since the computed ΔV does only depend on γ_0, γ_{deo}, r_0, r_{deo} and V_0 a test function has to check whether the correct entry interface is reached after the de-orbit burn was carried out. At $t = 1$ the spacecraft's orbital velocity is $\vec{V}_0$ and after the de-orbit maneuver the vehicle re-enters the atmosphere at the point indicated by the cross. Although the correct FPA γ_{deo} is found at the corresponding altitude of the entry interface, the de-orbit burn can not be initiated at $t = 1$, as the current position does not match the latitude and/or longitude of the entry interface. In consequence the spacecraft continues to fly its initial orbit. At $t = 2$, however, the entry interface can be reached with a de-orbit burn, and thus the test function can send the command to carry out the de-orbit maneuver.***

A final remark is to be made on the variables in equation (6.6): All variables of flight represent here absolute and not relative values. Thus, they do not correspond directly to those of the reference frame usually used within the computations. However, this approach provides a good first order approximation for the computation of a ΔV. A more detailed analysis of de-orbit maneuvers as well as the derivation of equation (6.6) is provided in *[8]*.

3) spacecraft_input.txt

While the previous input files contain information concerning the trajectory, this input file is used for the characterization of the spacecraft. A list of its parameters can be found in Table 6.5. For the computation of a launch trajectory the mass of the spacecraft is computed automatically during the execution of the program as it does not remain constant and changes quickly. Thus, while using *launch_input.txt* any value for the mass specified here in *spacecraft_input.txt* will not be considered. For the aerodynamic coefficients C_L and C_D two options are possible. Either they are specified in this file and they remain constant during

Parameter	Meaning [Unit]
m	Mass $[kg]$
S	Reference surface $[m^2]$
N_{C_L}	Lift Coefficient Flag:
	1) C_L is constant and specified hereafter
	2) C_L is varying and defined in an extra file
C_L	Lift coefficient (if specified via flag)
N_{C_D}	Drag Coefficient Flag:
	1) C_D is constant and specified hereafter
	2) C_D is varying and defined in an extra file
C_D	Drag coefficient (if specified via flag)
μ	Bank angle $[deg]$
α	Angle of attack $[deg]$

Table 6.5: ***Input parameters of spacecraft_input.txt***

the whole computation or they are read from additional files (see next section) as functions of the Mach number and the angle of attack (see 4.1.1.4). In the present version of the code the values for the reference surface S, for the bank angle μ and for the angle of attack α remain constant during the execution of the program.

4) CD_input.txt , CL_input.txt

These files contain the variables and parameters specified in Table 6.6 and provide the possibility to compute varying aerodynamic coefficients depending on the Mach number M and the angle of attack α. In addition to different values for the Mach number and for the AoA

Varible/Parameter	Meaning [Unit]
N_{mach}	Number of Mach numbers
N_{AoA}	Number of angles of attack
$M_1 \dots M_N$	Different Mach numbers
$\alpha_1 \dots \alpha_N$	Different angles of attack $[deg]$
C_D or C_L	Value for C_D or C_L for each specified combination of Mach number and AoA

Table 6.6: ***Input variables and parameters of CD_input.txt and CL_input.txt***

these files contain values for the aerodynamic coefficients for each specified combination of M and α. During the execution of the program this information is used to interpolate linearly between the specified values of C_D and C_L. Thus, dynamic and flight regime dependent values of C_D and C_L are obtained.

5) pitch_input.txt

This file is only mandatory for the computation of a launch trajectory. The local pitch angle ϑ_L was introduced in chapter 4. It is the angle between the local horizon and the spacecraft

x-axis. The pitch angle is used to lead the launcher on its way from the ground to the desired injection parameters as the rocket follows a pitch law preprogrammed in its guidance system. In *pitch_input.txt* the required parameters for the specification of a pitch law are provided. Table 6.7 shows the parameters contained in this file. The concrete values of those depend strongly on the launch configuration.

Parameter	**Meaning [Unit]**
N_{pitch}	Pitch Flag:
	1) Local pitch law
	2) Inertial pitch law
N_{eq}	Number of functions following
For each function:	
t_i	Start time $[sec]$
t_f	Stop time $[sec]$
ϑ_i	Pitch angle at start time $[deg]$
ϑ_f	Pitch angle at stop time $[deg]$

Table 6.7: ***Input parameters of pitch_input.txt***

The Pitch Flag indicates whether the data refers to a local (spacecraft linked) coordinate system (see Figure 4.1), or to a so-called Galilean or inertial (launch pad linked) coordinate system, as the pitch angle can be either measured with respect to the local horizon of the spacecraft or with respect to the local horizon of the launch pad. However, it is possible to transform the pitch angle from one system to the other. In particular for an equatorial launch this can be achieved by using

$$\vartheta_L(t) = \vartheta_I(t) + \frac{X(t)}{r_{ea}} + \Omega \cdot t \quad . \tag{6.7}$$

Here, $\vartheta_L(t)$ is the local pitch angle linked to the spacecraft, $\vartheta_I(t)$ represents the pitch angle measured from the launch pad, R_{ea} is the radius of the Earth, Ω its angular velocity and $X(t)$ represents the spacecraft's horizontal distance from the launch pad. The horizontal distance can be expressed via

$$\frac{dX}{dt} = r_{ea}\frac{V\cos\gamma}{r} \quad , \tag{6.8}$$

where r, V and γ represent the radius vector, the relative velocity and the flight path angle known from the set of differential equations.

The information shown in Table 6.7 is used to compute the pitch angle as a function of time. For each time interval specified in the file a function is created where $\vartheta_L(t)$ (or $\vartheta_I(t)$) is approximated by a straight. Thus, for each time step during the launch a value of ϑ_L (or ϑ_I) can easily be computed.

6) physical_model_input.txt

Several physical parameters have to be quantified or defined in this input file. Furthermore models for the atmosphere and the gravitational field can be selected. The different parameters contained in this file are listed in Table 6.8. Besides the radius and the angular velocity

Parameter	**Meaning [Unit]**
R_{ea}	Earth radius $[m]$
Ω	Angular velocity of the Earth $[rad/s]$
μ_{ea}	Gravitational parameter of the Earth $[m^3/s^2]$
UT	Universal time $[s]$
DOY	Day of year
$YEAR$	Actual year
N_{atmos}	Atmosphere Model Flag: 1) No atmosphere 2) Analytical model 3) NRLMSISE-00 model
N_{gravi}	Gravitational Field Flag: 1) Spherical gravitational field 2) J-2 Correction terms
dt	Integration time step $[s]$
N_{print}	Print Flag: Defines how frequently the results are written in the output files

Table 6.8: ***Input parameters of physical_model_input.txt***

of the Earth, the gravitational parameter has to be specified as well. For the trajectories presented in the next chapter the following values for these parameters are assumed:

$$R_{ea} = 6378136.49\,m$$

$$\Omega = 7.2921159 \cdot 10^{-5}\,rad/s$$

$$\mu_{ea} = 3.98941 \cdot 10^{14}\,m^3/s^2$$

The initial date and the time at which the trajectory computation is started has also to be chosen. This information is especially needed if the NRLMSISE-00 atmosphere model is selected via the atmosphere model flag, as here the atmospheric density and temperature depend on the time and the date (see 5.1.3). The gravitational field flag indicates whether the J-2 correction terms of the gravitational field shall be considered or not. If so, the gravitational pull of the Earth does not only depend on the altitude but also on the latitude and longitude (see 5.3). If the correction terms are neglected the gravitational acceleration of the Earth is

$$g(r) = \frac{\mu_{ea}}{r^2} \quad , \tag{6.9}$$

with r being the radius vector measured from the center of the Earth and μ_{ea} representing the gravitational parameter. Taking the values from above leads to a mean acceleration on the Earth surface of

$$g_0 = \frac{\mu_{ea}}{R_{ea}^2} \approx 9.8067\,m/s^2 \tag{6.10}$$

Finally, the length of the integration time step and the value of the Print Flag have to be quantified. The actual value of the integration time step might depend on the type of

trajectory to be computed, as re-entry and launch trajectories require normally a smaller time step than for instance the computation of circular orbits. However, the smaller the time step, the more accurate the computation but the longer the computation time.

By using the Print Flag it can be specified how frequently the results of the numerical integration are written in the output files. For example, this could be done after every iteration step ($N_{print} = 1$) or, preferably in case of long orbit propagations where the amount of data needs to be reduced, after every twentieth iteration step ($N_{print} = 20$).

7) stop_condition_input.txt

Table 6.9 shows the two parameters contained in this last input file. The first one is the so-called Type Flag N_{stop_type} specifying the type of the stop condition for the main loop of the program. For example the flag could indicate that the integration loop is stopped when a certain altitude is reached or, alternatively, when a certain time period has elapsed. Table 6.9 explains all options for this flag in detail.

The second parameter contains the limit of the loop itself. This could be, for example, an altitude of 200 km above the Earth surface or a time period of 1000 seconds.

Specifying the final condition for the loop in an extra input file offers more flexibility as the source code does not need to be modified.

Parameter	**Meaning [Unit]**
N_{stop_type}	Type Flag: It indicates whether the loop limit is the 1) Minimum altitude during descent 2) Maximum altitude during ascent 3) Amount of completed orbits 4) Time period of computation
X_{limit}	Value the of limit $[km]/[s]$/[INTEGER]

Table 6.9: ***Input parameters of stop_condition_input.txt***

6.1.2 The output files

After having discussed the input files in detail, the attention is now on the output files and the information they contain. In the actual program architecture three output files have been implemented.

1) Results.txt

This output file contains the evolution of the six variables of motion describing the trajectory. The variables provided in this file are shown in Table 6.10. One should note that the actual altitude of the spacecraft is not a direct output variable but has to be calculated. This arises due to the fact that the radius vector $\vec{r}$ is measured from the center of the Earth and not from its surface. Consequently, the Earth radius has to be subtracted from the value of r used during the computation in order to obtain the altitude above the Earth surface.

Variable	Meaning [Unit]
$YEAR$	Actual year
DOY	Day of year
UT	Universal time $[s]$
t	Elapsed trajectory time $[s]$
$r(t)$	Altitude above Earth's surface $[km]$
$V(t)$	Relative velocity $[m/s]$
$\gamma(t)$	Flight path angle $[deg]$
$L(t)$	Longitude $[deg]$
$l(t)$	Latitude $[deg]$
$\chi(t)$	Azimuth of relative velocity $[deg]$

Table 6.10: ***Output variables of Results.txt***

2) Results2.txt

The second output file contains information concerning the aerothermodynamic properties of the trajectory. Table 6.11 shows the related variables. The values for the air density, the temperature, the Mach number and the aerodynamic coefficients refer to the actual position of the spacecraft.

Variable	Meaning [Unit]
$YEAR$	Actual year
DOY	Day of year
UT	Universal time $[s]$
t	Elapsed trajectory time $[s]$
$\rho(r,t)$	Actual air density $[kg/m^3]$
$\Theta(r,t)$	Actual temperature $[K]$
M	Mach number
$C_D(M,\alpha)$	Drag coefficient
$C_L(M,\alpha)$	Lift coefficient

Table 6.11: ***Output variables of Results2.txt***

3) Results3.txt

The third and last output file contains information especially interesting for launch and re-entry trajectories. Table 6.12 summarizes the variables presented in this file. Besides the normal and tangential forces F_N and F_T, the evolution of the thrust and the mass are also provided. In addition, the so-called g-loads in tangential and normal direction in units of the Earth acceleration g_0 are given. They are of special interest for the structural stability of the spacecraft and can be computed via

$$g_{load,N}(t) = \frac{F_N(t)}{m(t)} \cdot \frac{1}{g_0} \ , \quad g_{load,T}(t) = \frac{F_T(t)}{m(t)} \cdot \frac{1}{g_0} \quad . \tag{6.11}$$

Variable	Meaning [Unit]
$YEAR$	Actual year
DOY	Day of year
UT	Universal time $[s]$
t	Elapsed trajectory time $[s]$
$F_N(t)$	Normal force $[N]$
$F_T(t)$	Tangential force $[N]$
$g_{load,N}(t)$	Normal g-loads $[g_0]$
$g_{load,T}(t)$	Tangential g-loads $[g_0]$
$m(t)$	Mass $[kg]$
$T(t)$	Thrust $[N]$

Table 6.12: ***Output variables of Results3.txt***

6.1.3 Program structure

After having presented the input and output files in the previous pages the general data flow and the data processing of the developed program is subject of this section. A graphical representation is given in Figure 6.2 where the general program structure is shown in a flowchart.

It was mentioned that three major input files define the type of trajectory to be computed (6.1.1.1). Thus, by selecting the input file the initial values of the six variables of motion are selected and defined as well. Before running the main loop (i.e. before starting the numerical integration of the six variables) the information concerning the spacecraft characteristics (e.g. mass, surface, etc.) has to be read from the relevant input files. As mentioned previously, in case of a launch trajectory the mass (like the thrust) is provided by the main input file and not by the spacecraft input file. It consists of the sum of payload mass, propellant mass, structural mass (separation mass) and additional mass items. Once the initial configuration is found the numerical loop starts its iterations. At each time step within this process different functions provide the actual data of the atmospheric conditions, of the gravitational field and, if required, the actual values of the aerodynamic coefficients. For a launch trajectory the actual thrust, provided by the ignited engines, is computed as well. Again, the mass consists of the actual sum of the four contributions mentioned above. By combining all information the resulting forces acting on the spacecraft can be calculated and the subsequent values of the six variables of motion can be predicted. Additionally, new values for the g-loads and the Mach number can also be computed.

As pointed out in 6.1.1.2 it is possible to initiate a de-orbit maneuver in order to re-enter the atmosphere and to reach a specified entry interface. In this case, the program checks within the loop whether a de-orbit maneuver, when it was initiated instantaneously, would lead to the entry conditions specified in the input file or not. If so, the velocity will be automatically decreased by the computed ΔV (6.6) and the spacecraft will start its descent.

Depending on the Print Flag (Table 6.8) the computed values of all variables are written in the different output files. These provide a wide range of information concerning the trajectory and the spacecraft characteristics.

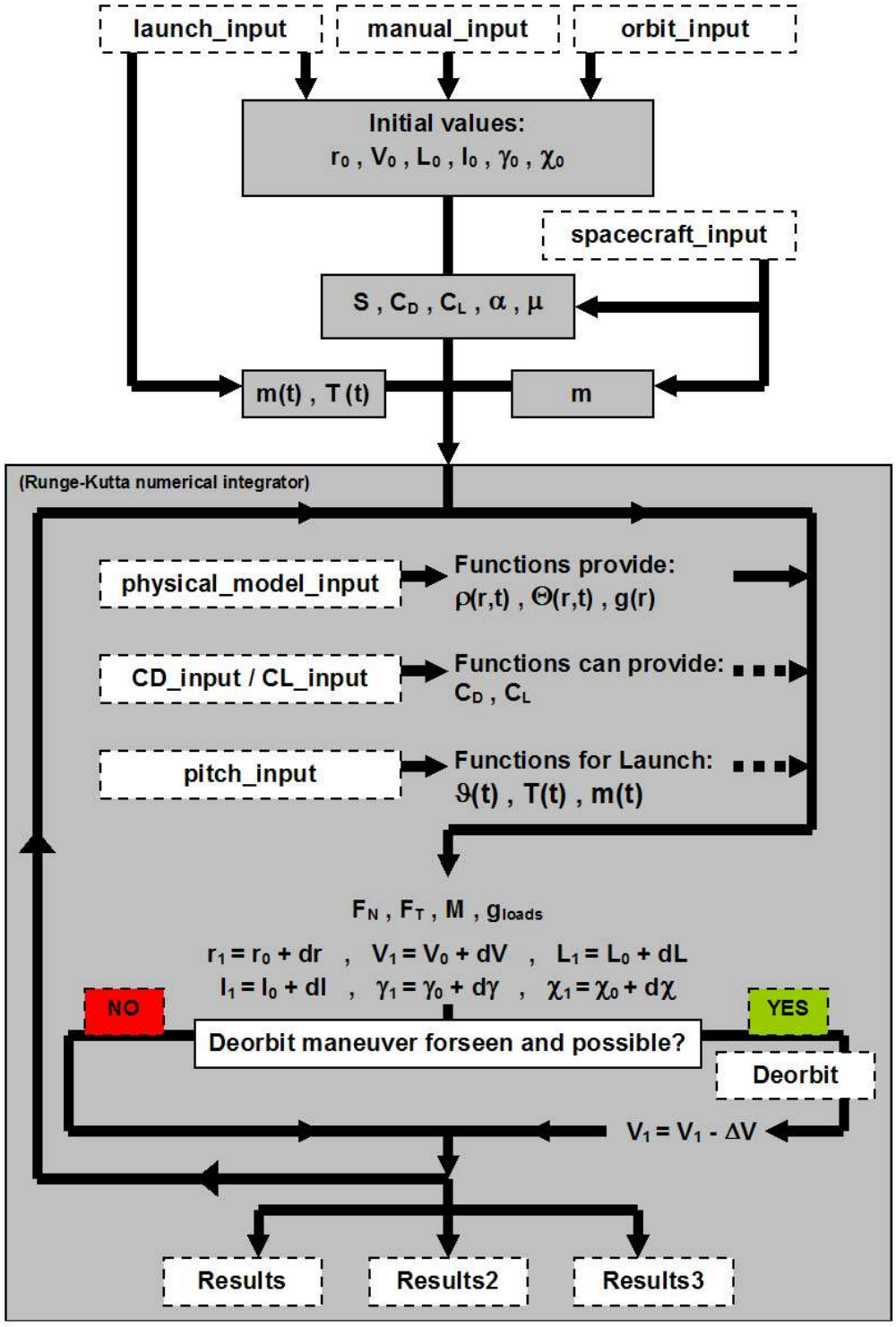

Figure 6.2: ***Flowchart showing the general structure of the software program as explained in section 6.1.3. The upper part defines the initial conditions and the spacecraft properties, the lower part (contained in the box) represents the loop of the numerical integration.***

6.2 The numerical scheme: The Runge-Kutta algorithm

Since the numerical integrator scheme plays a key role in the computer program it is discussed here in greater detail. In *[14]* numerous examples for numerical integrators with different advantages and characteristics can be found.

Due to its stability and accuracy the numerical scheme used in this work is an 8^{th}-order Runge-Kutta algorithm. From the very common 4^{th}-order version of this approach, explained in detail in the following, the 8^{th}-order algorithm can be derived easily. The 4^{th}-order approach was also investigated but it proved to be insufficient in the sense of stability and accuracy. Therefore the higher order algorithm was selected for the final implementation.

The problem to solve is a so-called initial value problem of ordinary differential equations. Given the function

$$\frac{dy}{dx} = f(x,\, y) \tag{6.12}$$

and a readily available point $(x_n,\, y_n)$ (solution of the equation above) a subsequent solution can be found by using the linear approximation

$$(x_n,\, y_n) \quad \Rightarrow \quad x_{n+1} = x_n + h, \;\; y_{n+1} = y_n + hf(x_n,\, y_n). \tag{6.13}$$

This scheme was already developed by Euler and carries therefore his name. The selected width of the step h depends strongly on the problem. However, this method is not very accurate and suitable for functions varying significantly as its error is of the order of $\mathcal{O}(h^2)$.

The Runge-Kutta approach is quite similar but benefits from the fact that it does not use the slope at the initial point but a corrected value. The order of this correction corresponds to the order of the Runge-Kutta scheme and shows also in the order of the error. The 4^{th}-order Runge-Kutta version has an error of the order of $\mathcal{O}(h^5)$ and is based on the following algorithm:

$$k_1 = hf(x_n\,,\, y_n) \tag{6.14}$$

$$k_2 = hf(x_n + \frac{h}{2}\,,\, y_n + \frac{k_1}{2}) \tag{6.15}$$

$$k_3 = hf(x_n + \frac{h}{2}\,,\, y_n + \frac{k_2}{h}) \tag{6.16}$$

$$k_4 = hf(x_n + h\,,\, y_n + k_3) \tag{6.17}$$

$$x_{n+1} = x_n + h \tag{6.18}$$

$$y_{n+1} = y_n + \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4) + \mathcal{O}(h^5) \tag{6.19}$$

This algorithm is frequently used for numerical integrations as it is easy to implement and offers, in most cases, satisfying stability. Furthermore, it can also be applied to systems of differential equations. In this case every component of the system is advanced simultaneously and explicitly. Having the following set of differential equations

$$\dot{y}^{(1)} = f(t\,,\, y^{(1)},\, y^{(2)})$$
$$\dot{y}^{(2)} = g(t\,,\, y^{(1)},\, y^{(2)})$$

and the initial values $(t_n\,,\, y_n^{(1)}\,,\, y_n^{(2)})$, one can calculate the subsequent points by using

$$k_0^{(1)} = hf(t_n\,,\, y_n^{(1)}\,,\, y_n^{(2)}) \qquad k_0^{(2)} = hg(t_n\,,\, y_n^{(1)}\,,\, y_n^{(2)})$$

$$k_1^{(1)} = hf(t_n + \frac{1}{2}h\,,\, y_n^{(1)} + \frac{1}{2}k_0^{(1)}\,,\, y_n^{(2)} + \frac{1}{2}k_0^{(2)}) \qquad k_1^{(2)} = hg(t_n + \frac{1}{2}h\,,\, y_n^{(1)} + \frac{1}{2}k_0^{(1)}\,,\, y_n^{(2)} + \frac{1}{2}k_0^{(2)})$$

$$k_2^{(1)} = hf(t_n + \frac{1}{2}h\,,\, y_n^{(1)} + \frac{1}{2}k_1^{(1)}\,,\, y_n^{(2)} + \frac{1}{2}k_1^{(2)}) \qquad k_2^{(2)} = hg(t_n + \frac{1}{2}h\,,\, y_n^{(1)} + \frac{1}{2}k_1^{(1)}\,,\, y_n^{(2)} + \frac{1}{2}k_1^{(2)})$$

$$k_3^{(1)} = hf(t_n + h\,,\, y_n^{(1)} + k_2^{(1)}\,,\, y_n^{(2)} + k_2^{(2)}) \qquad k_1^{(2)} = hg(t_n + h\,,\, y_n^{(1)} + k_2^{(1)}\,,\, y_n^{(2)} + k_2^{(2)})$$

$$y_{n+1}^{(1)} = y_n^{(1)} + \frac{1}{6}k_0^{(1)} + \frac{1}{3}k_1^{(1)} + \frac{1}{3}k_2^{(1)} + \frac{1}{6}k_3^{(1)} \tag{6.20}$$

$$y_{n+1}^{(2)} = y_n^{(2)} + \frac{1}{6}k_0^{(2)} + \frac{1}{3}k_1^{(2)} + \frac{1}{3}k_2^{(2)} + \frac{1}{6}k_3^{(2)} \tag{6.21}$$

The only difference between the 4^{th}-order Runge-Kutta algorithm and the 8^{th}-order version is the higher accuracy of the second scheme. It has an error of only $\mathcal{O}(h^9)$. By using the following general notation

$$k_j = f(t_n + c_j h\,,\, y_n + h\sum_{k=0}^{j-1} a_{jk}k_k) \tag{6.22}$$

$$y_{n+1} = y_n + \sum_{j=0}^{j_{max}} b_j\, k_j \tag{6.23}$$

the 4^{th}-order Runge-Kutta scheme is characterized by the coefficients shown in Table 6.13. For the 8^{th}-order algorithm Table 6.14 summarizes the corresponding coefficients.

j	c_j	$a_{j\,k},\ k=$	0	1	2	b_j
0	0					$\frac{1}{6}$
1	$\frac{1}{2}$		$\frac{1}{2}$			$\frac{1}{3}$
2	$\frac{1}{2}$		0	$\frac{1}{2}$		$\frac{1}{3}$
3	1		0	0	1	$\frac{1}{6}$

Table 6.13: ***Coefficients of the*** 4^{th}***-order Runge-Kutta algorithm.***

j	c_j	$a_{jk},\ k=$	0	1	2	3	4	5	6	b_j
0	0									$\frac{7}{1408}$
1	$\frac{1}{6}$		$\frac{1}{6}$							0
2	$\frac{4}{15}$		$\frac{4}{75}$	$\frac{16}{75}$						$\frac{1125}{2816}$
3	$\frac{2}{3}$		$\frac{5}{6}$	$-\frac{8}{3}$	$\frac{5}{2}$					$\frac{9}{32}$
4	$\frac{4}{5}$		$-\frac{8}{5}$	$\frac{144}{25}$	-4	$\frac{16}{25}$				$\frac{125}{768}$
5	1		$\frac{361}{320}$	$-\frac{18}{5}$	$\frac{407}{128}$	$-\frac{11}{80}$	$\frac{55}{128}$			0
6	0		$-\frac{11}{640}$	0	$\frac{11}{256}$	$-\frac{11}{160}$	$\frac{11}{256}$	0		$\frac{5}{66}$
7	1		$\frac{93}{640}$	$-\frac{18}{5}$	$\frac{803}{256}$	$-\frac{11}{160}$	$\frac{99}{256}$	0	1	$\frac{5}{66}$

Table 6.14: ***Coefficients of the*** 8^{th}***-order Runge-Kutta algorithm.***

In chapter 8 the performance and the accuracy of the 8^{th}-order Runge-Kutta algorithm is discussed in detail and the results obtained with this scheme are compared to those of other numerical integrators commonly used in astrophysical computations.

In the following pages chapter 7 shows numerous examples of trajectories all computed with the program presented in this chapter. It is demonstrated that a unified approach to the computation of spacecraft trajectories is possible, that it provides good results for all flight phases and that it proves to be a solid basis for enhanced future applications.

Chapter 7

The Results: Trajectories

In this chapter different examples of spacecraft trajectories are presented. They illustrate the feasibility of a general treatment of trajectories as they are all computed with the program presented in the previous chapter. The examples cover all types of trajectories and flight phases mentioned so far, i.e. satellite orbits as well as re-entry and launch trajectories. The correctness of the methodology is demonstrated by validating and comparing the results to those from other sources and the available literature. Before more complex trajectories are discussed, simple examples are shown at the beginning. Initially these were used to check on the general performance of the software tool.

7.1 Closed orbits

In this section four examples of closed orbits are presented. Two circular orbits (not inclined and inclined) are shown at the beginning followed by a highly eccentric orbit (a so-called Molniya orbit). Finally, the results from a re-computation of the orbit decays of the STARSHINE1 and STARSHINE2 satellites complete this section.

7.1.1 Circular orbit (1)

This first example represents a circular orbit lying in the equatorial plane with an altitude of $r = 1000$ km. The gravitational field of the Earth is assumed to be spherically symmetric and the atmosphere is neglected. Hence, this orbit is a completely unperturbed Keplerian orbit.

In Figures 7.1 - 7.6 the evolution of the six variables of motion is shown for a period of six orbits. Due to the orbit parameters and the physical model configuration all variables are supposed to remain constant, except for the longitude. This behavior is clearly confirmed by the plots. The integration time step is chosen to be 10 seconds for this example. However, even with a time step of about 100 seconds, corresponding to approximately 70 integration steps per orbit, the plots look very much the same.

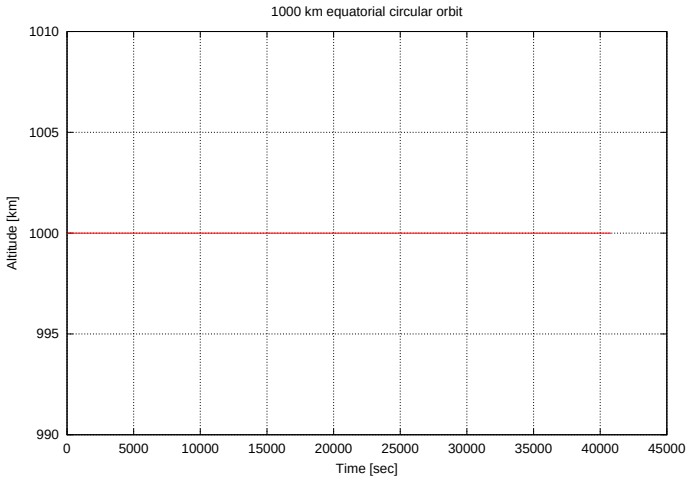

Figure 7.1: ***Circular Orbit (1): Altitude vs. Time***

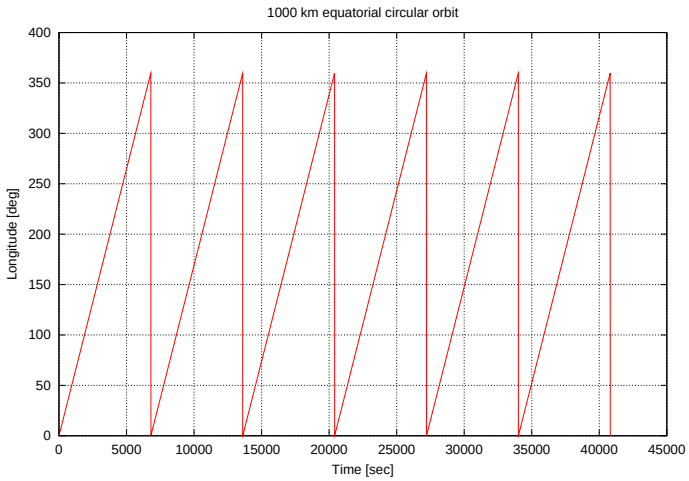

Figure 7.2: ***Circular Orbit (1): Longitude vs. Time***

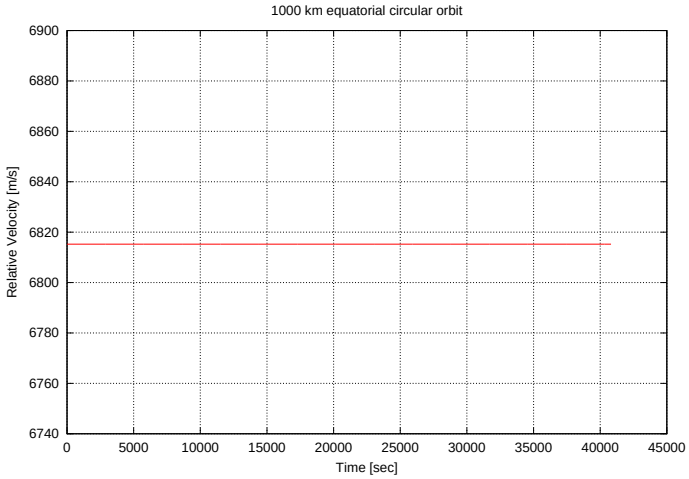

Figure 7.3: ***Circular Orbit (1): Relative Velocity vs. Time***

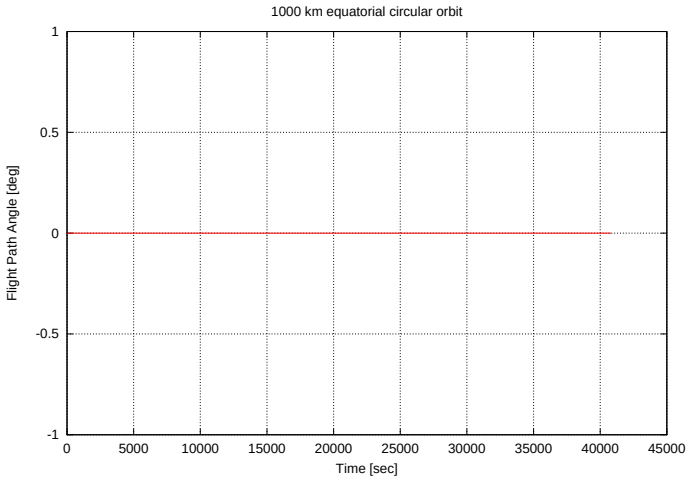

Figure 7.4: ***Circular Orbit (1): Flight Path Angle vs. Time***

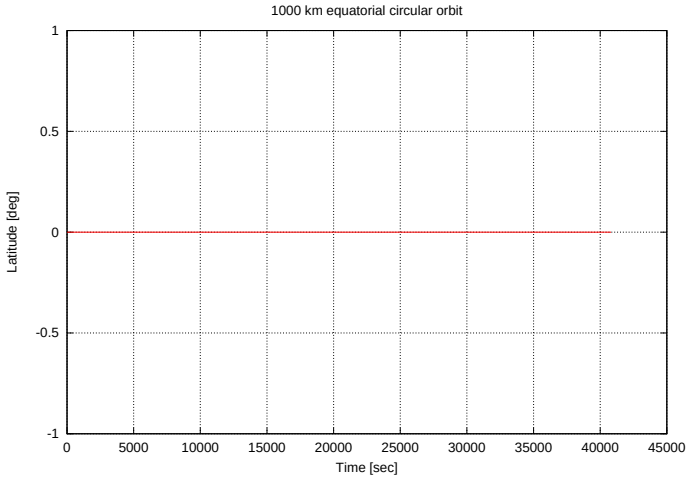

Figure 7.5: ***Circular Orbit (1): Latitude vs. Time***

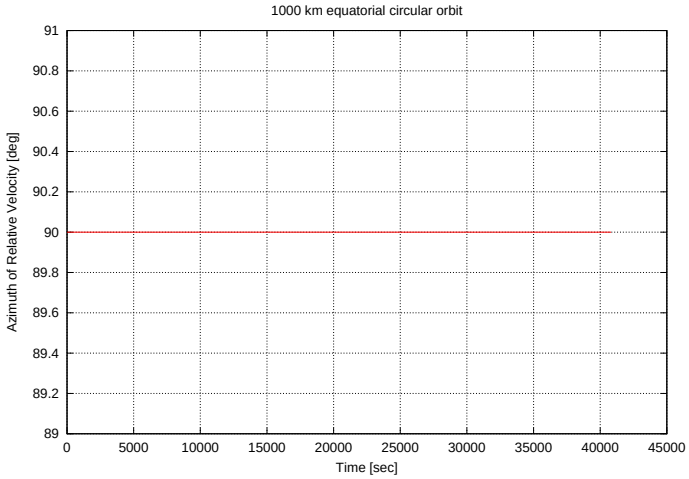

Figure 7.6: ***Circular Orbit (1): Azimuth of the Relative Velocity vs. Time***

7.1.2 Circular orbit (2)

Figures 7.7 - 7.12 show a circular orbit with an altitude of $r = 400$ km and an inclination of $i = 51.6°$. This corresponds approximately to the orbit of the International Space Station (ISS). Like in the previous example the gravitational field is assumed to be spherically symmetric and the atmosphere is neglected. Therefore no variation in the altitude, no orbit decay is observed (Figure 7.7) and the FPA remains also constant (Figure 7.10). However, due to the inclination of the orbit some of the other variables of motion show a different behavior than in the previous example. The relative velocity varies over a restricted range and reaches its maximum when the spacecraft passes the equator (Figure 7.9). The azimuth of the relative velocity and the latitude do not remain constant either. Figure 7.11 reveals that the azimuth always reaches a value of $\chi = 90°$ when the latitude passes through its maximum or minimum value. Referring to the definition of the azimuth in Figure 3.2 this behavior is expected. Figure 7.12 shows the typical ground-track pattern of an inclined circular orbit. The plots are based on computations with an integration time step of 10 seconds.

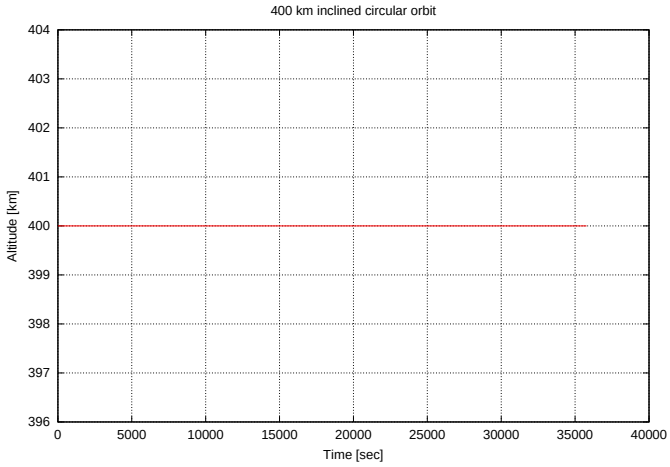

Figure 7.7: ***Circular Orbit (2): Altitude vs. Time***

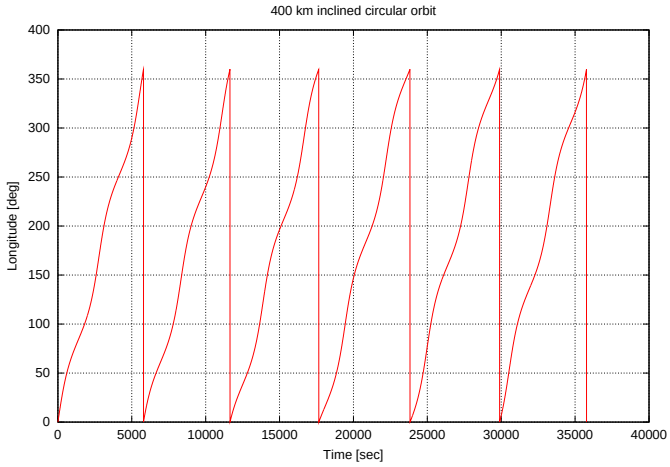

Figure 7.8: ***Circular Orbit (2): Longitude vs. Time***

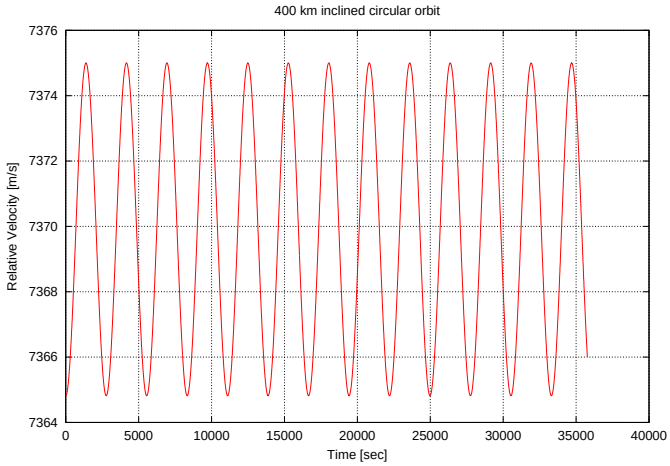

Figure 7.9: ***Circular Orbit (2): Relative Velocity vs. Time***

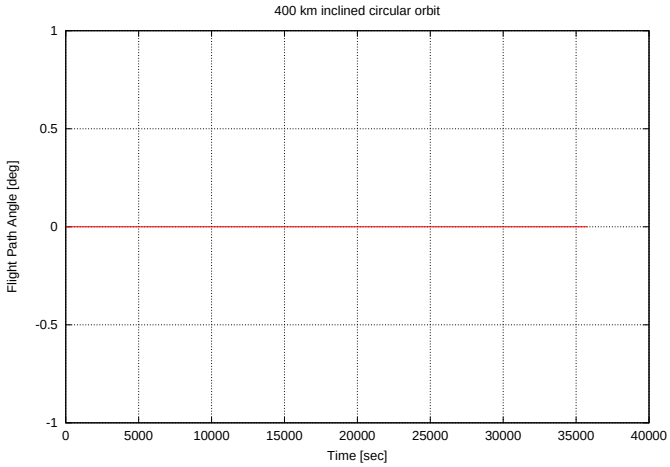

Figure 7.10: ***Circular Orbit (2): Flight Path Angle vs. Time***

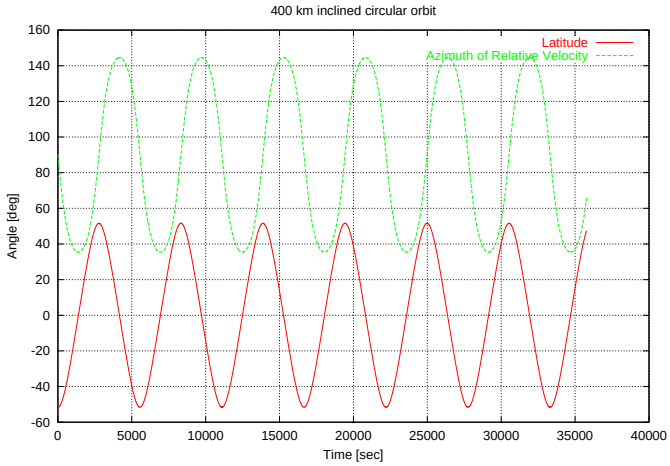

Figure 7.11: ***Circular Orbit (2): Latitude and Azimuth of the Relative Velocity vs. Time***

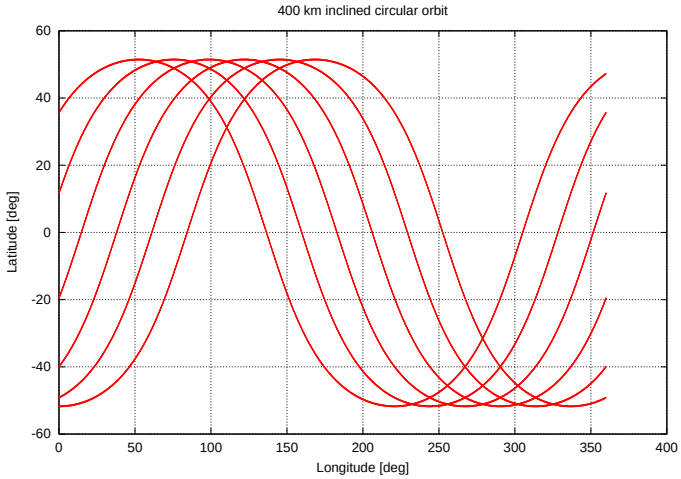

Figure 7.12: ***Circular Orbit (2): Latitude vs. Longitude***

7.1.3 Molniya orbit

The so-called Molniya orbits are inclined orbits with a high eccentricity and are frequently used for Russian telecommunication satellites. Figures 7.13 - 7.18 show the results of the computation for a typical Molniya orbit configuration. The apoapsis is set to $r_{apo} = 39360$ km while the periapsis is only $r_{peri} = 1000$ km. The inclination of the orbit is $i = 63.4°$. No atmosphere is considered and the gravitational field is assumed to be spherically symmetric. The integration time step is 10 seconds. Due to the high eccentricity of the orbit the resulting plots look different than those of the previous sections. The relative velocity varies immensely and reaches a minimum value close to zero in the apoapsis (Figure 7.14). The FPA does not remain constant but varies over wide range of values (Figure 7.16). Figure 7.17 confirms that the azimuth is $\chi = 90°$ whenever the latitude passes through its minimum or maximum. The computed ground-track depicted in Figure 7.18 is very similar to that of the reference trajectory shown in Figure 7.19. The latter one was taken from *[7]* for comparison.

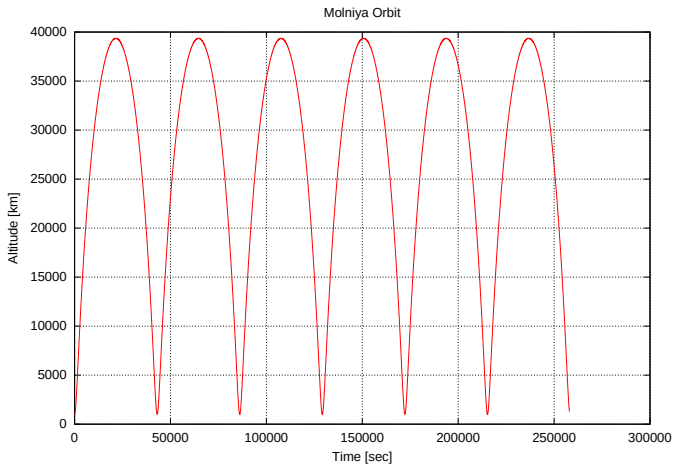

Figure 7.13: ***Molniya Orbit: Altitude vs. Time***

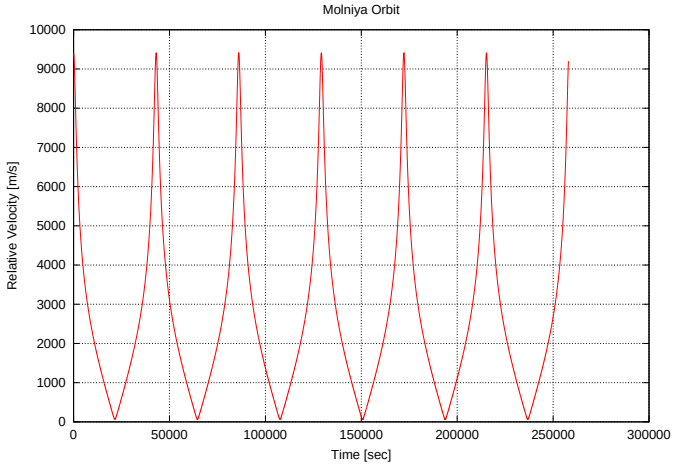

Figure 7.14: ***Molniya Orbit: Relative Velocity vs. Time***

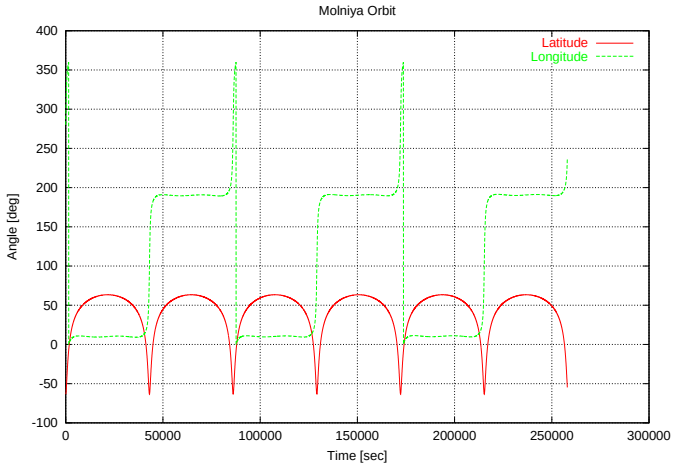

Figure 7.15: ***Molniya Orbit: Latitude and Longitude vs. Time***

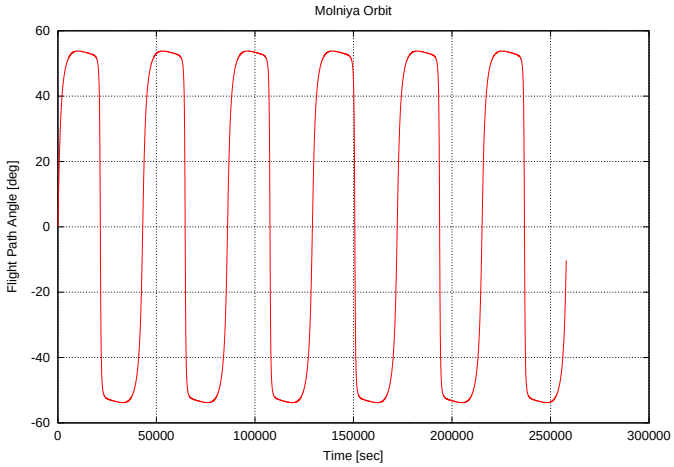

Figure 7.16: ***Molniya Orbit: Flight Path Angle vs. Time***

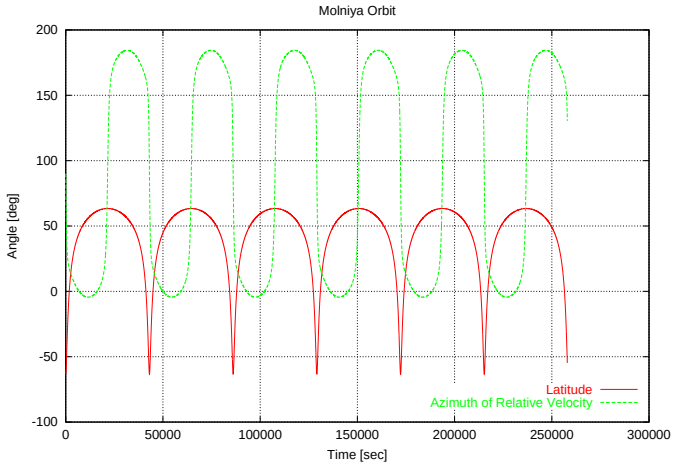

Figure 7.17: ***Molniya Orbit: Latitude and Azimuth of the Relative Velocity vs. Time***

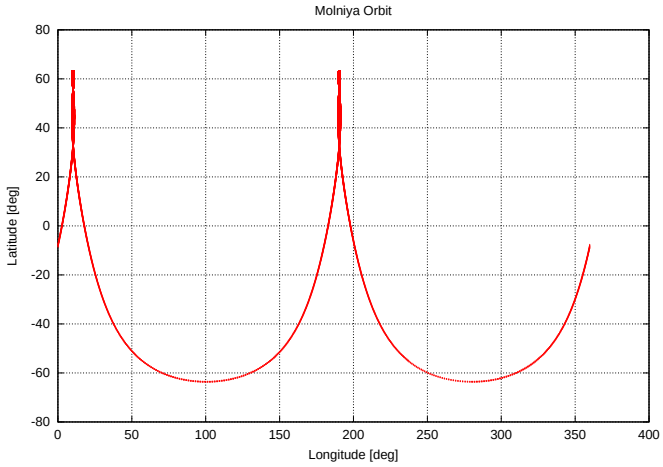

Figure 7.18: ***Molniya Orbit: Latitude vs. Longitude***

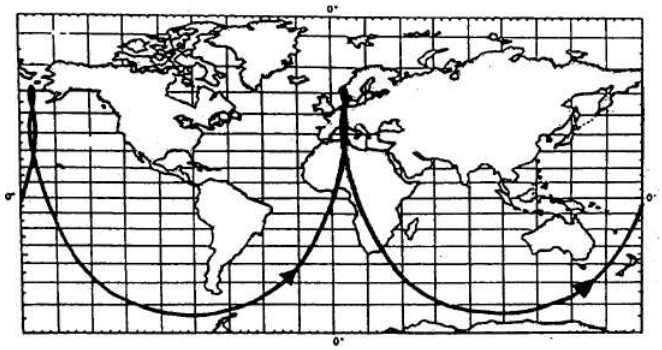

Figure 7.19: ***Reference ground-track for a typical Molniya orbit. The computed results shown in Figure 7.18 match the track depicted here very well. (Taken from [7] for comparison)***

7.1.4 STARSHINE1 and STARSHINE2 orbit decay

STARSHINE1 was a spherical satellite that was released from the payload bay of a Space Shuttle Orbiter in mid 1999 (see Figure 7.20). Due to the effect of atmospheric drag the

Figure 7.20: ***The STARSHINE1 satellite is released from the payload bay of a Space Shuttle Orbiter. (Taken from [26])***

initially circular orbit decayed and STARSHINE1 burned in the atmosphere during the reentry. The orbit decay was carefully surveyed from Earth as by relating these observations to computed trajectories precious information about the consistency of atmospheric and exoatmospheric environmental models should be obtained. Since the initial conditions and the satellite configuration are known the orbit decay can be recomputed. The results can then be

Parameter	**Value**
Mass	39 kg
Reference surface	0.181 m^2
Drag coefficient C_D	2.1
Lift coefficient C_L	0.0
Initial altitude	387 km
Inclination	51.6°
Initial relative velocity	$\sim$ 7367.71 m/s
Initial latitude (chosen)	$-51.6°$
Initial longitude (chosen)	200°
DOY	156
YEAR	1999

Table 7.1: ***Parameters of the STARSHINE1 satellite. The values for the initial latitude and longitude are chosen arbitrarily as other values were not accessible.***

compared to the published flight date analysis. The parameters shown in Table 7.1 are used as input. In addition, the J2-correction terms of the gravitational field and a slightly modified

NRLMSISE-00 atmosphere model are required for a realistic reproduction of the trajectory.[1]

Besides the trajectory of STARSHINE1 also the orbit decay of the STARSHINE2 satellite can be recomputed. STARSHINE2 followed the same approach and served the same purpose like its precursor. It had the same shape and spacecraft parameters (e.g. mass, reference surface) but was launched 18 months later. The initial altitude was approximately 17 km lower than that of STARSHINE1. The computation of the STARSHINE2 orbit decay is based on the initial conditions shown in Table 7.2.

Parameter	**Value**
Initial altitude	370 km
Initial relative velocity	$\sim$ 7378.04 m/s
DOY	350
YEAR	2001

Table 7.2: ***Parameters of the STARSHINE2 satellite. The other parameters concerning the initial orbit and the spacecraft configuration equal those of STARSHINE1 shown in Table 7.1.***

Figure 7.21 and Figure 7.23 show the results of the re-computations. The integration time step was set to 10 seconds. The faster orbit decay of the second satellite can be explained by a lower initial altitude and very high solar activity at the beginning of 2002 (Figure 5.3). Figure 7.22 and Figure 7.24 represent the results of the flight data analysis. They were taken from *[9]* and *[13]* for comparison. Since the computed results match very well the reference plots it is shown that with the presented unified approach and the developed software tool even long term predictions with high accuracy are possible.

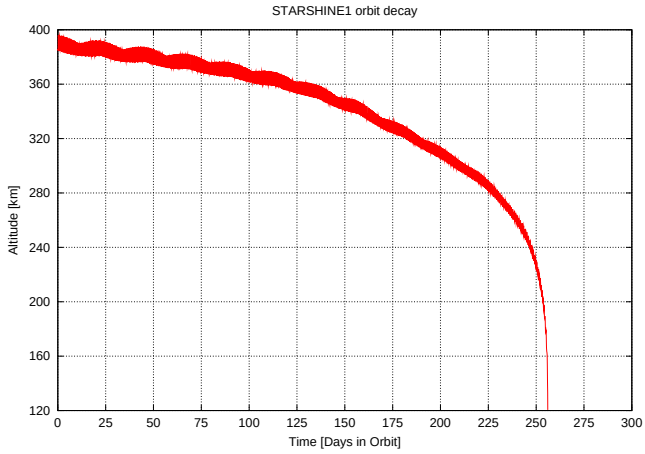

Figure 7.21: ***STARSHINE1 orbit decay computation: Altitude vs. Time. The entry interface at 120 km is reached after approximately 256 days.***

[1]The STARSHINE1 orbit decay was partly used to check the quality of the NRLMSISE-00 atmosphere model. The analysis of the measurements revealed that the computed total mass density is, on average, over-predicted. Therefore the atmospheric density calculated by NRLMSISE-00 has been corrected by a factor of 0.85 for all computations.

Figure 7.22: ***STARSHINE1 orbit decay as computed from flight data analysis [9]. The entry interface at 120 km is reached after approximately 259 days.***

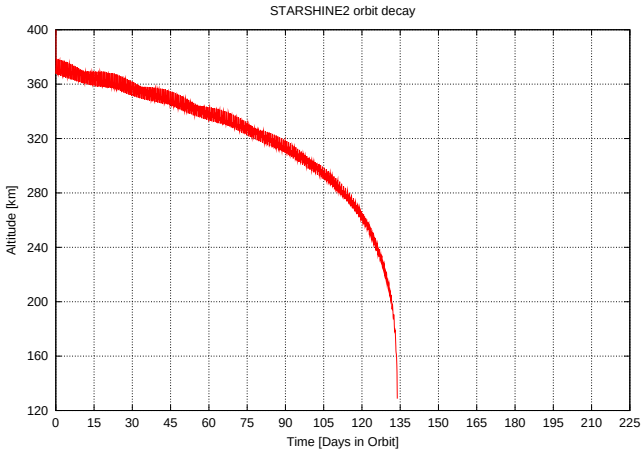

Figure 7.23: ***STARSHINE2 orbit decay computation: Altitude vs. Time. The entry interface at 120 km is reached after approximately 134 days.***

Figure 7.24: ***STARSHINE2 orbit decay as computed from flight data analysis [13]. The entry interface at 120 km is reached after approximately 131 days.***

7.2 Re-entry trajectories

In this section three examples for re-entry trajectories are presented. At the beginning two computations are shown that were used to validate the program. Their results are compared to those of accessible literature. Thereafter, the computation of a ballistic re-entry of a Soyuz capsule coming from the International Space Station (ISS) is discussed.

7.2.1 Generic re-entry trajectory (1)

The following plots correspond to the computation of a generic re-entry trajectory starting at $r_0 = 120$ km and going down to ground level. The initial relative velocity is $V_0 = 7850$ m/s and the initial flight path angle is $\gamma_0 = -5°$. As the flight is supposed to take place in the equatorial plane the initial latitude is set to $L_0 = 0°$. For the initial longitude a value of $l_0 = 0°$ is chosen arbitrarily. The atmosphere is represented by the analytical model and the gravitational field is assumed to be spherically symmetric. The spacecraft has a mass of $m = 10000$ kg and a reference surface of $S = 100$ m^2. The aerodynamic coefficients are constantly $C_D = 0.5$ and $C_L = 0.1$. The bank angle is kept at $\mu = 0°$ during the whole descent. Since the variables of motion change more rapidly during the re-entry compared to the orbiting phase, the integration time step is set to 1 second. The computed results can be used to validate the program as this trajectory is also discussed in *[8]*.

Figure 7.25 shows the altitude profile of the trajectory. Due to its aerodynamic properties and the initial conditions the spacecraft loses constantly height, except for a small interval between 130 and 180 seconds. In this interval the flight path angle is no longer negative as it can be seen in Figure 7.29. Figure 7.26 shows the altitude as a function of the relative velocity for which Figure 7.27 is the reference plot whose flight profile is matched very well. Figure 7.28 depicts again the relative velocity, but now as a function of time. Besides a small increase in the velocity at the beginning, the spacecraft is decelerated rapidly due to the influence of the atmosphere. Figure 7.30 confirms that the flight takes place in the equatorial plane as the latitude remains 0° and the azimuth of the relative velocity is constantly 90°.

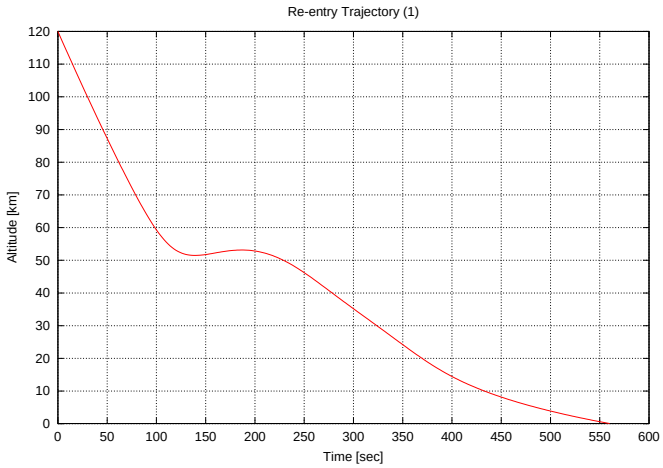

Figure 7.25: ***Generic re-entry Trajectory (1): Altitude vs. Time***

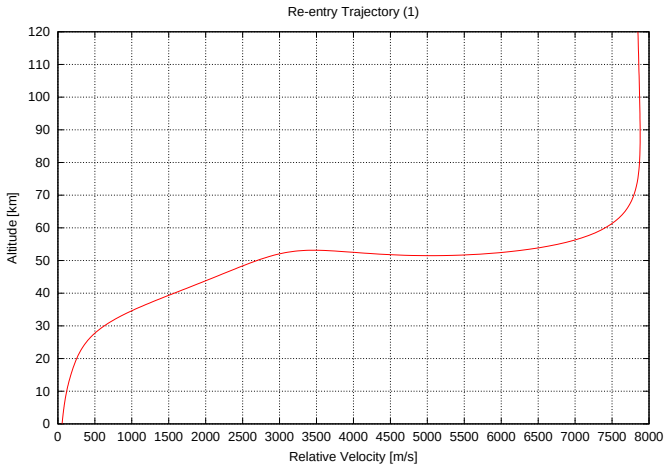

Figure 7.26: ***Generic re-entry Trajectory (1): Altitude vs. Relative Velocity***

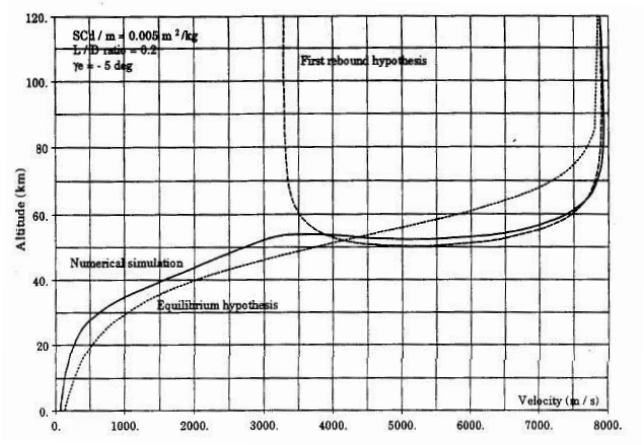

Figure 7.27: ***This plot is taken from [8] for comparison and shows the altitude as a function of the relative velocity. The computed results of Figure 7.26 correspond to the curve with the index "Numerical Simulation".***

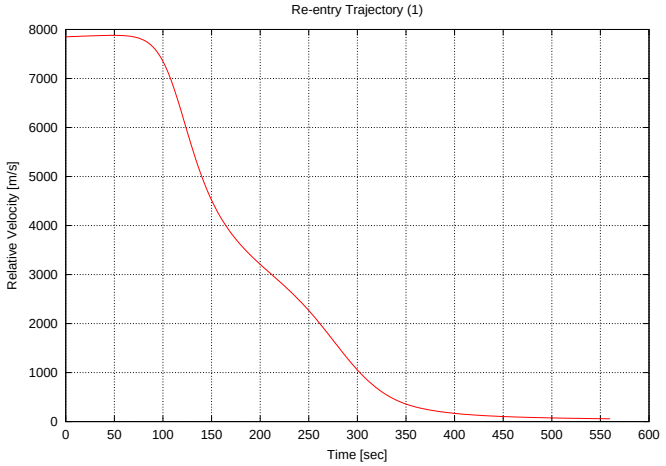

Figure 7.28: ***Generic re-entry Trajectory (1): Relative Velocity vs. Time***

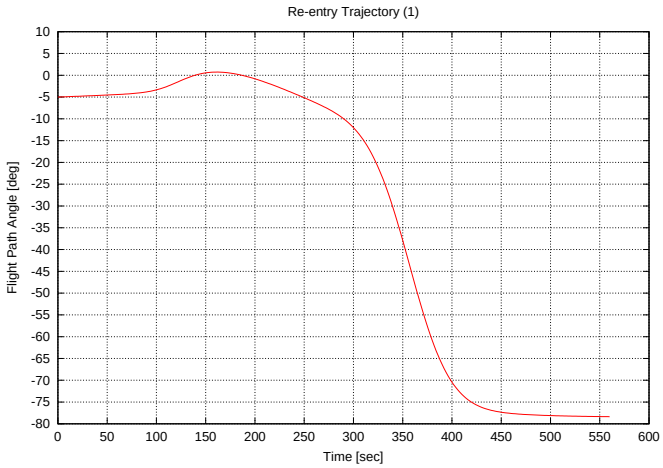

Figure 7.29: ***Generic re-entry Trajectory (1): Flight Path Angle vs. Time***

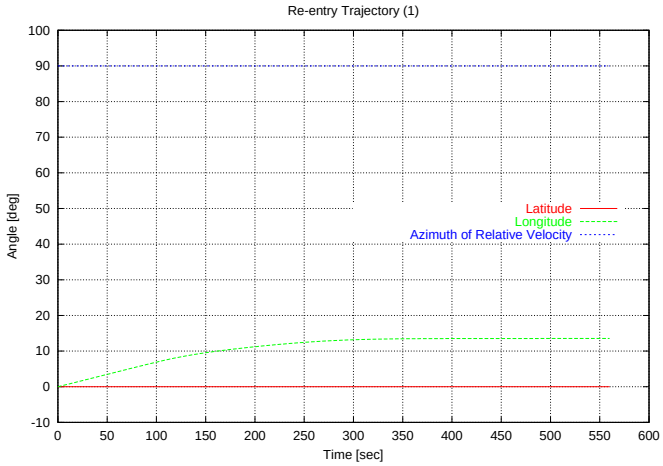

Figure 7.30: ***Generic re-entry Trajectory (1): Latitude, Longitude and Azimuth of the Relative Velocity vs. Time***

7.2.2 Generic re-entry trajectory (2)

This example is similar to the previous one and is also taken from *[8]* for comparison and validation purposes. However, some of the initial conditions are chosen differently. The initial relative velocity for the following plots is $V_0 = 7500$ m/s, the initial flight path angle is $\gamma_0 = -1.3°$ and the lift coefficient is $C_L = 0.5$ during the whole re-entry. Although the other variables have the same values like in the previous example, the resulting trajectory looks very different, as Figures 7.31, 7.32 and 7.34 - 7.36 clearly demonstrate. Special interest shall be paid to the fact that in this configuration it takes much longer for the spacecraft to reach the ground which results from numerous atmospheric rebounds the vehicle encounters (Figure 7.31). Figure 7.32 shows the altitude as a function of the relative velocity and must be compared to Figure 7.33 being the reference plot. As one can see, the computed results are in very good agreement with those taken from *[8]*.

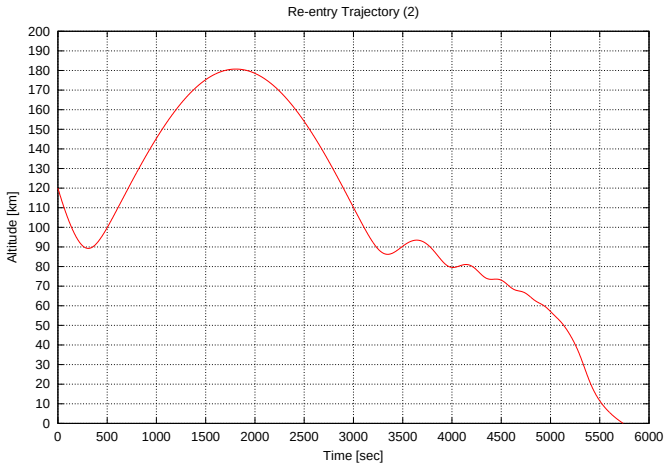

Figure 7.31: ***Generic re-entry Trajectory (2): Altitude vs. Time***

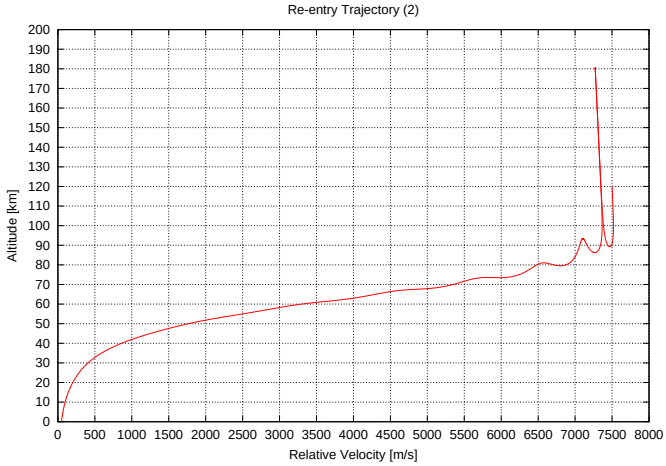

Figure 7.32: ***Generic re-entry Trajectory (2): Altitude vs. Relative Velocity***

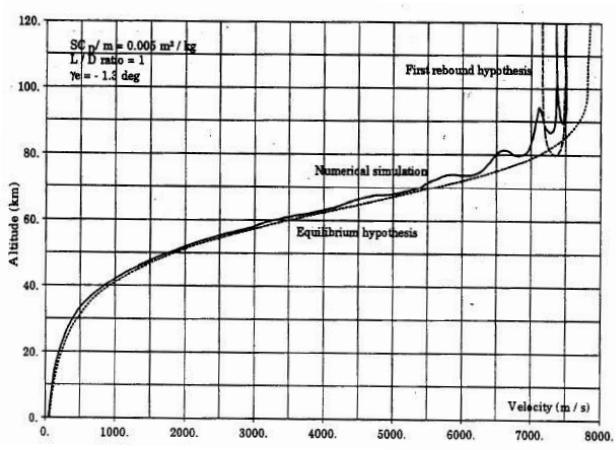

Figure 7.33: ***This plot is taken from [8] for comparison and shows the altitude as a function of the relative velocity. The computed results of Figure 7.32 correspond to the curve with the index "Numerical Simulation".***

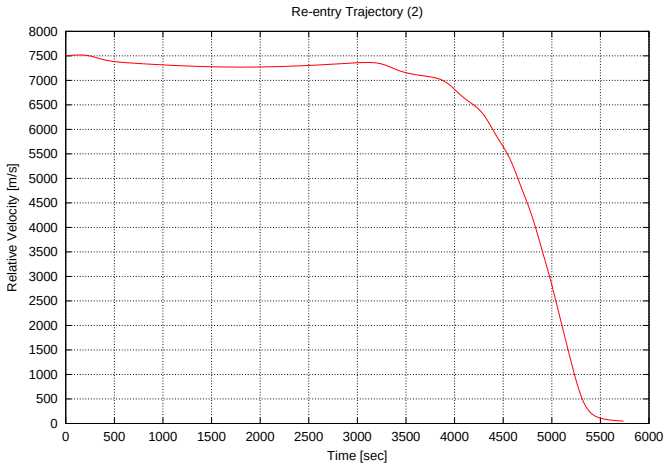

Figure 7.34: ***Generic re-entry Trajectory (2): Relative Velocity vs. Time***

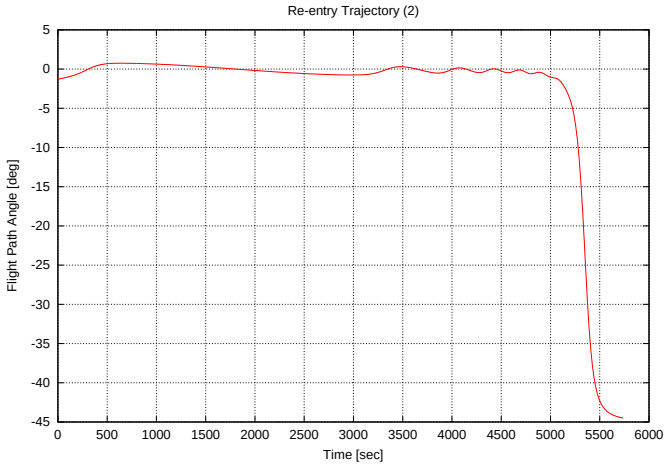

Figure 7.35: ***Generic re-entry Trajectory (2): Flight Path Angle vs. Time***

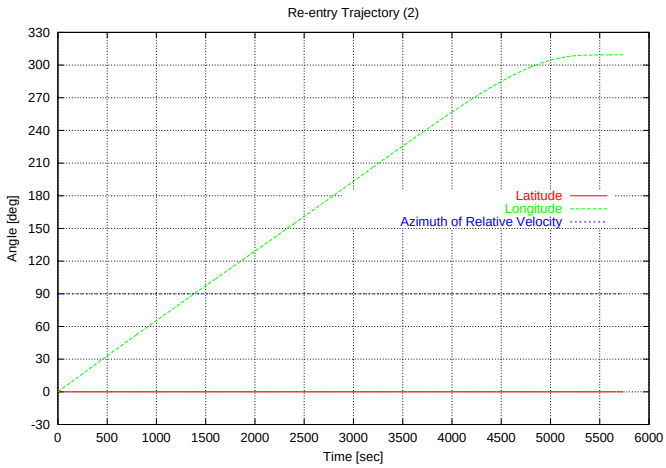

Figure 7.36: ***Generic re-entry Trajectory (2): Latitude, Longitude and Azimuth of the Relative Velocity vs. Time***

7.2.3 Soyuz capsule: Ballistic re-entry coming from the ISS

After having shown two re-entry trajectories going from entry interface down to ground level, the following example includes the orbiting phase and a de-orbit maneuver. Since the ballistic re-entry of a Soyuz capsule coming from the ISS is simulated, this is a more realistic case than the previous generic examples.

The initial orbit is circular, has an altitude of $r = 390$ km and an inclination of $i = 51.6°$. The analytical atmospheric model is chosen and the gravitational field of the Earth is assumed to be spherically symmetric. In order to compute a de-orbit maneuver the entry interface has to be specified (see 6.1.1.2). In the present case the entry interface is defined by

$$l_{deo} = 42.266°, \quad L_{deo} = 40.431°, \quad r_{deo} = 102\,km\ , \quad \gamma_{deo} = -1.48° \quad .$$

This takes into account that the capsule is supposed to touch down in a foreseen area in Kazakhstan. Concerning the characteristics of the capsule the following values are assumed:

$$S = 3.8\,m^2, \quad C_D = 1.26\ , \quad C_L = 0.0\ , \quad m = 2900\,kg \quad .$$

With this information the program computes a ΔV leading the spacecraft from its initially circular orbit to the above specified entry interface. One should be reminded, that the above specified flight path angle γ_{deo} is an absolute flight path angle and not a flight path angle of the relative velocity as usually. The orbit is started at $t = 0$ seconds at $L = -51.6°$ latitude and $l = 279.16°$ longitude. The integration time step is set to 1 second.

Computed results	**Reference results *[19]***
De-orbit burn:	
$\Delta V = 114.154$ m/s	$\Delta V = 115.2$ m/s
Entry Interface Conditions:	
$r = 102.1$ km	$r = 101.783$ km
$l = 42.25°$	$l = 42.266°$
$L = 40.32°$	$L = 40.426°$
$V = 7610.3$ m/s	$V = 7620$ m/s
$\gamma = -1.54°$	$\gamma = -1.57°$
Parachute Deployment Conditions:	
$r = 10.7$ km	$r = 10.7$ km
$l = 68.38°$	$l = 69.0°$
$L = 49.78°$	$L = 50.08°$
$V = 242.95$ m/s	$V = 200.0$ m/s

Table 7.3: ***Comparison of the results for a Soyuz re-entry computation: The computed values are shown on the left while the reference values are shown on the right. They are provided by ESA [19].***

Figures 7.37 - 7.40 show the results of the computation. Table 7.3 provides a direct comparison between the computed results and reference values taken from *[19]*. The appearing differences might arise from different assumptions concerning the modelling of the aerodynamic

properties of the spacecraft. Nevertheless this example demonstrates that computations combining the orbiting and the re-entry phase and including a de-orbit maneuver can be treated with the actual formulation and that a change of reference frames is not required even if the trajectory includes atmospheric and exo-atmospheric parts.

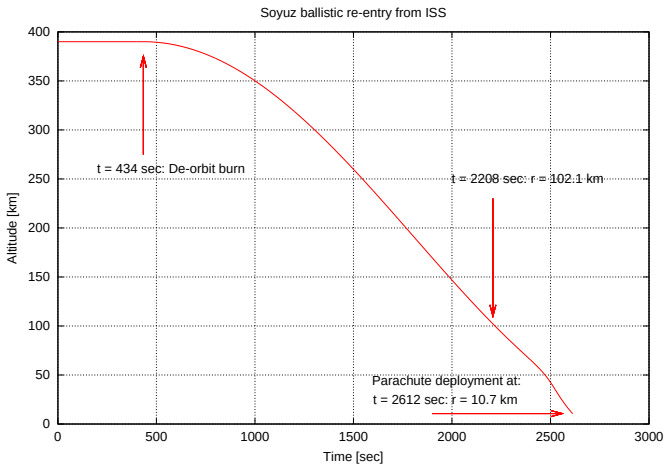

Figure 7.37: ***Soyuz re-entry coming from the ISS: Altitude vs. Time. The de-orbit burn initiates the descent of the spacecraft. 1774 seconds after the de-orbit maneuver the capsule passes the entry interface at 102.1 km. 404 seconds later at 10.7 km altitude the parachute of the capsule is deployed.***

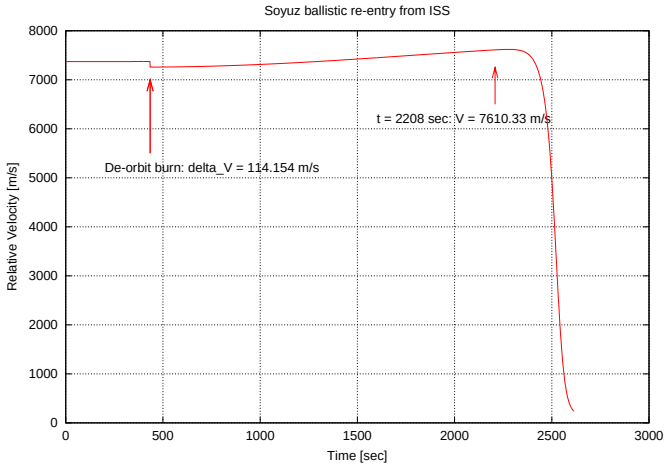

Figure 7.38: ***Soyuz re-entry coming from the ISS: Relative Velocity vs. Time. At $t = 434$ seconds the decrease in the relative velocity is clearly visible (de-orbit burn). At $t = 2208$ seconds the entry interface at 102.1 km is reached.***

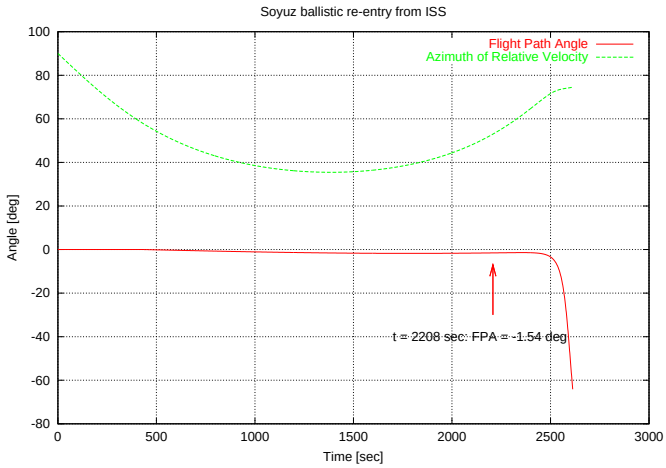

Figure 7.39: ***Soyuz re-entry coming from the ISS: Flight Path Angle and Azimuth of the Relative Velocity vs. Time. For the entry interface passage at $t = 2208$ seconds the value of the FPA is indicated. It matches very well the reference value as seen in Table 7.3.***

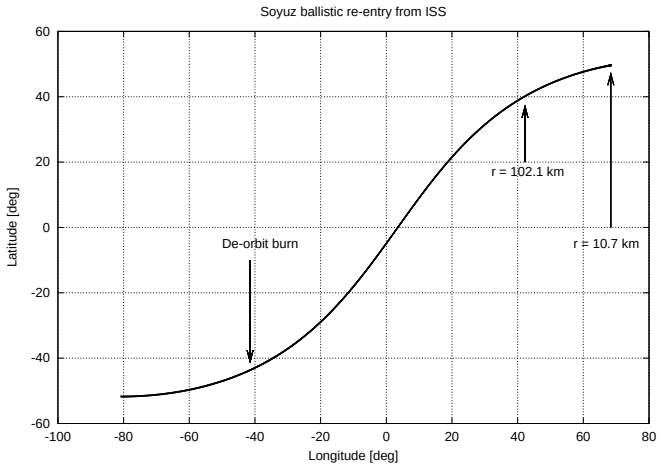

Figure 7.40: ***Soyuz re-entry coming from the ISS: Latitude vs. Longitude. The spacecraft's geographic positions during the main events (de-orbit burn, entry interface passage, parachute deployment) are indicated. The capsule touches down in Kazakhstan as expected.***

7.3 Launch trajectories

This section contains three examples of computed launch trajectories. The first one is the launch of an Ariane 44LP rocket assumed to take place in the equatorial plane. Thereafter, the results for two launches of a Soyuz rocket from the European launch pad in Kourou in French Guiana are shown. They differ slightly in the formulation of the pitch law and refer to reference cases presented in *[15]* and *[20]*. Before going deeper into the different examples some general remarks concerning launch trajectories shall be made.

As previously mentioned in chapters 4.2 and 6.1.1.5 a pitch law needs to be imposed during the ascent in order to lead the launcher on its way to space. However, it is very difficult to find detailed data for the formulation of this pitch law for any launch configuration. In most cases the pitch law has to be derived from plots as no analytical expressions are accessible. In addition to that, the values for the mass, the thrust, the specific impulse or the burn duration of an engine can vary significantly if one refers to different data sources. Some of the parameters are also mission dependent. All these inconsistencies lead to uncertainties in the computation of the trajectory.

Concerning the equations of motion a small modification has to be made for the computation of a launch: It has to be considered that as long as the pitch angle ϑ_L (or ϑ_I) is set to 90° (typically the first couple of seconds after lift off) the flight path angle γ remains 90° as well. From that it results, that for this period the differential equation describing changes in the azimuth of the relative velocity (3.35) needs to be disregarded in order to avoid a division by zero.

7.3.1 Ariane 44LP: Launch in the equatorial plane

Although a launch in the equatorial plane (i.e. launch pad sited on the equator and a target orbit with $i = 0°$) is not a very realistic case, this examples provides the possibility to compare the results to data from other sources and thus to validate the program. As mentioned above, the launcher configuration and the pitch law are crucial elements for the computation of the launch trajectory. Table 7.4 contains the pitch law used for the computation of this Ariane launch. In Figure 7.41 a graphical representation is given. This pitch law refers to a

Time period [s]	ϑ_I **at the beginning**	ϑ_I **at the end**
$0.0 < t < 12.0$	90.0°	90.0°
$12.0 < t < 25.0$	90.0°	89.5°
$25.0 < t < 30.0$	89.5°	88.0°
$30.0 < t < 40.0$	88.0°	86.0°
$40.0 < t < 100.0$	86.0°	51.8°
$100.0 < t < 130.0$	51.8°	29.4°
$130.0 < t < 150.0$	29.4°	22.8°
$150.0 < t < 540.0$	29.4°	−7.2°
$540.0 < t < 1082.0$	−7.2°	−58.0°
$1082.0 < t < 1100.0$	−58.1°	−61.0°

Table 7.4: ***Pitch law for the computation of the Ariane 44LP launch in the equatorial plane. The values of ϑ_I refer to a Galilean reference frame, i.e. they are measured with respect to the launch pad.***

Galilean reference frame, so that the pitch angle ϑ_I is measured relative to the launch pad. Figure 7.42 is taken from *[18]* and shows the pitch law that served as a basis for the derivation of Figure 7.41.

For an Ariane launcher different mission dependent booster configurations are possible. The Ariane 44LP version is a three stage rocket whose first stage consists of four main engines combined with two liquid and two solid boosters. Figure 7.50 shows an Ariane 44LP lifting off from French Guiana. The solid and liquid boosters attached to the main body are clearly

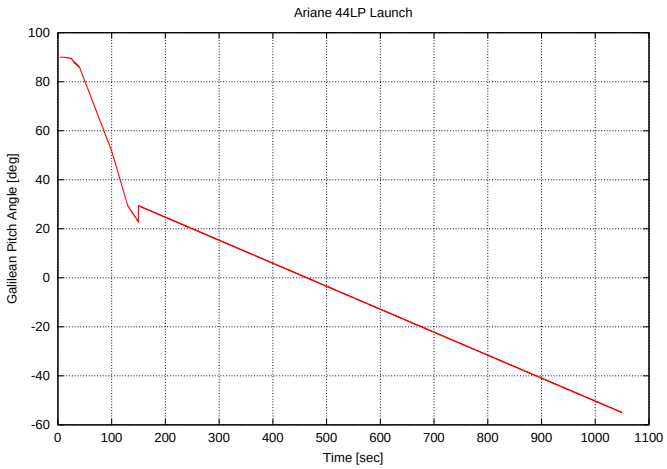

Figure 7.41: ***Ariane 44LP Launch Trajectory: Galilean Pitch Angle vs. Time. The plot depicts the pitch law defined in Table 7.4. This pitch law is used for the trajectory computation.***

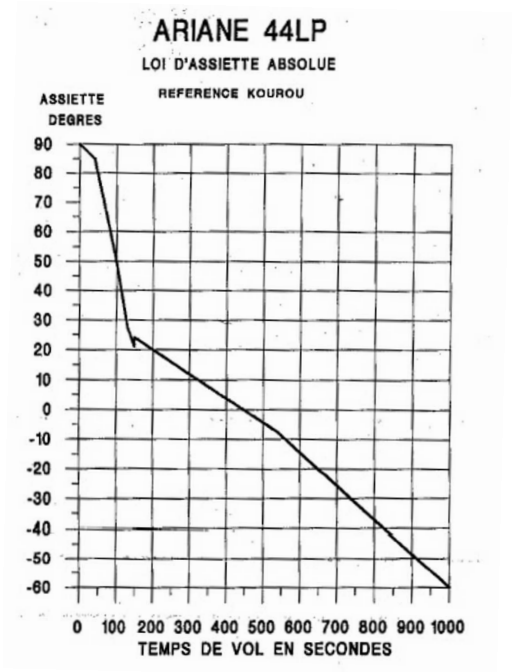

Figure 7.42: ***Reference pitch law for an Ariane 44LP launch. The Galilean Pitch Angle is shown as a function of time. This plot served as a basis for the derivation of the pitch law depicted in Figure 7.41 as an analytical expression of the pitch law for this Ariane 44LP launch is not accessible. (Taken from [18])***

visible. As pointed out previously, the characteristics of the boosters and the stages can vary if different data sources are considered.[2] The result of the computation here are based on the first stage configuration described in Table 7.5. Table 7.6 contains the values for the second and third stage.

First stage	
Main engine:	
Total thrust	4· 689150 N
Angle between thrust and spacecraft main axis	0°
Specific impulse	2432 m/s
Total propellant mass	233680 kg
Separation mass	17750 kg
Burn time	0-208.5 sec
Separation time	209.5 sec
Liquid boosters:	
Total thrust	2· 625000 N
Angle between thrust and spacecraft main axis	9°
Specific impulse	2404 m/s
Total propellant mass	79150 kg
Separation mass	8850 kg
Burn time	0-142 sec
Separation time	149 sec
Solid boosters:	
Total thrust	2· 670000 N
Angle between thrust and spacecraft main axis	12°
Specific impulse	2368 m/s
Total propellant mass	19000 kg
Separation mass	6200 kg
Burn time	3-39 sec
Separation time	66 sec

Table 7.5: ***Characteristics of the Ariane 44LP first stage booster combination as used for the trajectory computation.***

Besides the structural mass of the launcher and the propellant mass three additional mass items are considered for the trajectory computation. The payload mass is assumed to be 3195 kg. The fairing, jettisoned after 287 seconds, has a mass of 831 kg. And finally, 6000 kg of water are burned at a constant rate during the firing of the main engine in order to cool the nozzles. In that respect it is important to know, that the rocket lifts off 3.4 seconds after the ignition. That means that some fuel is already burned while the launcher is still on the launch pad.

With all this information, the mass and the thrust history of the launcher is determined. For the reference surface a value of $S = 23$ m^2 is assumed during the whole ascent phase. The

[2] In *[21]* and *[22]* descriptions of the most common launch systems can be found. One will notice that some of the figures differ significantly.

Second stage	
Total thrust	807800 N
Angle between thrust and spacecraft main axis	0°
Specific impulse	2870 m/s
Total propellant mass	35490 kg
Separation mass	3720 kg
Burn time	217-343 sec
Separation time	345 sec
Third stage	
Total thrust	63200 N
Angle between thrust and spacecraft main axis	0°
Specific impulse	4356 m/s
Total propellant mass	10838 kg
Separation mass	1780 kg
Burn time	350-1050 sec
Separation time	1100 sec

Table 7.6: ***Characteristics of the Ariane 44LP second and third stage boosters as used for the trajectory computation.***

lift coefficient C_L is set to zero while the drag coefficient C_D depends on the Mach number as described in Table 7.7. The values for the drag coefficient are taken from *[18]*. The integration time step for the computation is set to 0.1 seconds. The analytical atmosphere model is selected and a spherically symmetric gravitational potential of the Earth is assumed.

Mach Number	**Drag coefficient** C_d	**Mach Number**	**Drag coefficient** C_d
<0.3	0.90106	2.0	1.3728
0.49	0.90155	2.5	1.0076
0.51	0.75434	3.4	0.9008
0.9	0.99208	3.9	0.85
1.0	1.4322	4.1	0.68
1.12	1.9311	4.5	0.6044
1.3	1.9271	5.0	0.5863
1.55	1.7168	5.95	0.5568
1.7	1.5865	7.0	0.51189
		>12.0	0.5189

Table 7.7: ***The drag coefficient C_D of the Ariane 44LP launcher as a function of the Mach number. These values were taken from [18].***

Figure 7.43, 7.46 and 7.48 show the computed results obtained with specified input data. The Altitude-Time profile in Figure 7.43 looks very similar to those in Figure 7.44 and Figure 7.45 which are taken from *[16]* and *[18]* for comparison. However, one can see that the trajectories do not match perfectly. A possible reason was already mentioned: The results depend sensitively on the imposed pitch law and also on the mass and thrust history. Since

reliable values for these parameters, belonging to the trajectories shown in Figure 7.44 and 7.45, are not accessible, the flight profile can not be recomputed exactly. However, the presented plots reveal that the general characteristics of the trajectory are well reproduced. This hold true for the plots of the Altitude-Time profile already mentioned above, but also for those showing the Velocity-Time profile (Figure 7.46 and 7.47) and the Load Factor-Time profile (Figure 7.48 and 7.49).

Thus, this example confirms that launch trajectories can be computed with the same approach previously applied to the other flight phases.

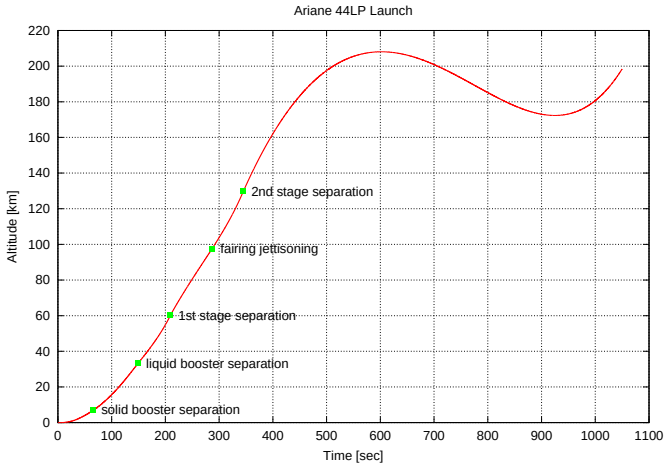

Figure 7.43: ***Ariane 44LP Launch Trajectory: Altitude vs. Time. The main events during the ascent (e.g. the separation of the different stages) are indicated. After approximately 600 seconds the rocket redescends in order to gain more velocity. This behavior as well as the final ascent after 920 seconds is controlled via the imposed pitch law.***

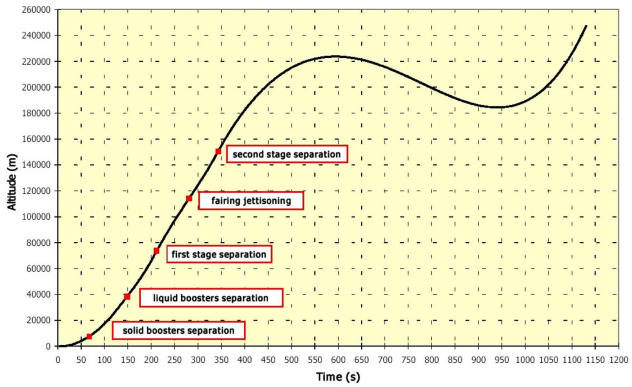

Figure 7.44: ***Reference trajectory (1) for an Ariane 44LP launch showing Altitude vs. Time. The main events during the ascent are indicated. (Taken from [16] for comparison)***

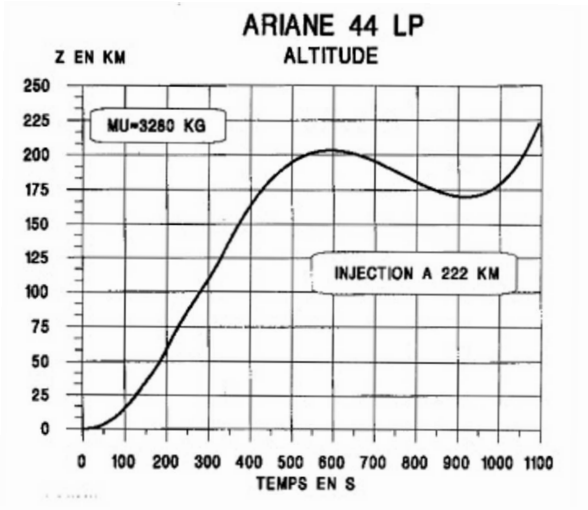

Figure 7.45: ***Reference trajectory (2) for an Ariane 44LP launch showing Altitude vs. Time. (Taken from [18] for comparison)***

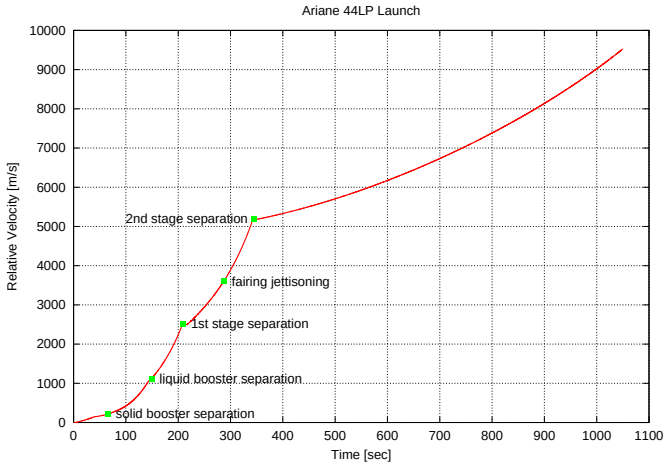

Figure 7.46: ***Ariane 44LP Launch Trajectory: Relative Velocity vs. Time***

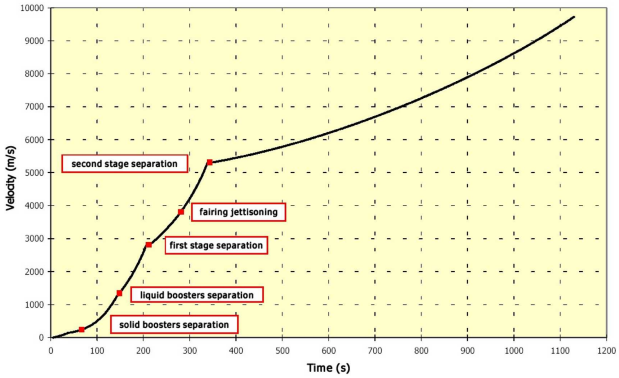

Figure 7.47: ***Reference trajectory (1) for an Ariane 44LP launch showing Relative Velocity vs. Time. (Taken from [16] for comparison)***

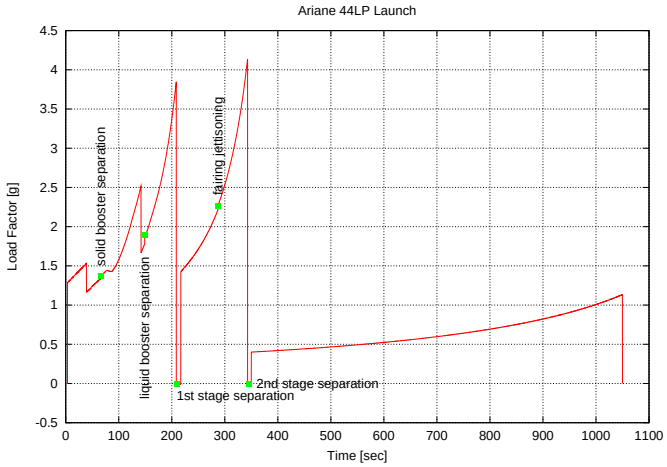

Figure 7.48: ***Ariane 44LP Launch Trajectory: Load Factor vs. Time***

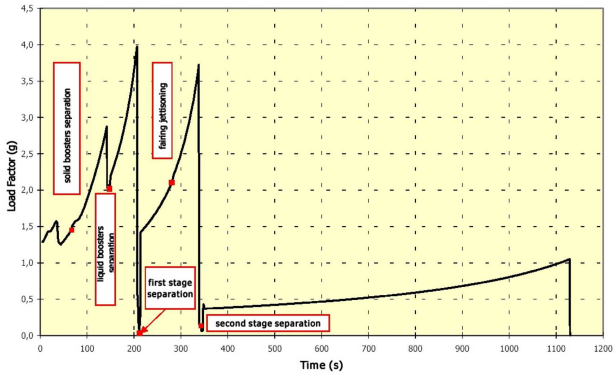

Figure 7.49: ***Reference trajectory (1) for an Ariane 44LP launch showing Load Factor vs. Time. (Taken from [16] for comparison)***

Figure 7.50: ***An Ariane 44LP lifting off from a launch pad in French Guiana. The solid and liquid boosters attached to the main body are clearly visible. (Courtesy of ESA)***

7.3.2 Soyuz: Launch from Kourou (1)

This example does not only consider another launch system than the previous one but it also represents a more realistic trajectory: A Soyuz rocket is launched from Kourou in French Guiana aiming for a transfer orbit with an inclination of $i = 51.6°$. This corresponds to the inclination of the orbit of the ISS. Compared to the example of the next section (7.3.3) the only difference is the imposed pitch law. Thus, the influence of the pitch angle will become more transparent, as the mass, the thrust and other parameters are the same for both examples. The pitch law used for the computation of the first Soyuz launch trajectory is shown in Table 7.8. A graphical representation is given in Figure 7.8. In order to derive this pitch law, Figure 7.52 that was taken from *[15]* served as a reference.

Time period [s]	ϑ_I **at the beginning**	ϑ_I **at the end**
$0.0 < t < 25.0$	$90.0°$	$90.0°$
$25.0 < t < 45.0$	$90.0°$	$80.0°$
$45.0 < t < 92.0$	$80.0°$	$49.5°$
$92.0 < t < 129.0$	$49.5°$	$37°$
$129.0 < t < 143.0$	$37.0°$	$30°$
$143.0 < t < 285.0$	$30.0°$	$11.1°$
$285.0 < t < 300.0$	$11.1°$	$11.1°$
$300.0 < t < 308.0$	$11.1°$	$13.3°$
$308.0 < t < 540.0$	$13.3°$	$-10.0°$

Table 7.8: ***Pitch law for the computation of a Soyuz launch from Kourou as derived from Figure 7.52. The values refer to a Galilean reference frame, i.e. they are measured with respect to the launch pad.***

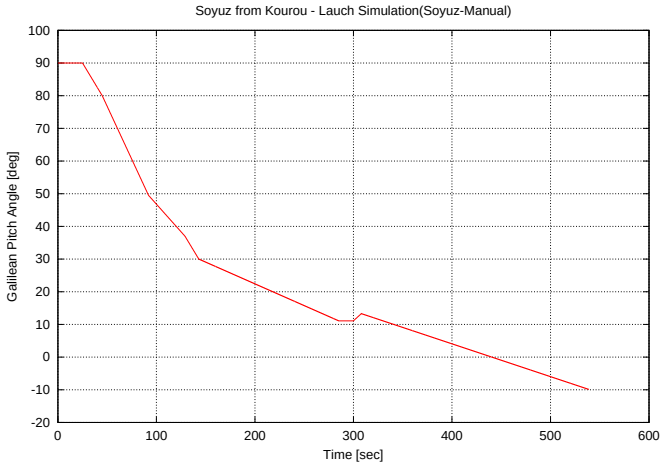

Figure 7.51: ***Soyuz Launch Trajectory (1): Galilean Pitch Angle vs. Time. The plot corresponds to the data contained in Table 7.8.***

Like the Ariane 44LP the Russian Soyuz rocket is a three stage rocket. The first stage consists also of four engines but it does not have any additional boosters. At the end of the following section Figure 7.70 shows a Soyuz rocket during lift off. Table 7.9 summarizes the

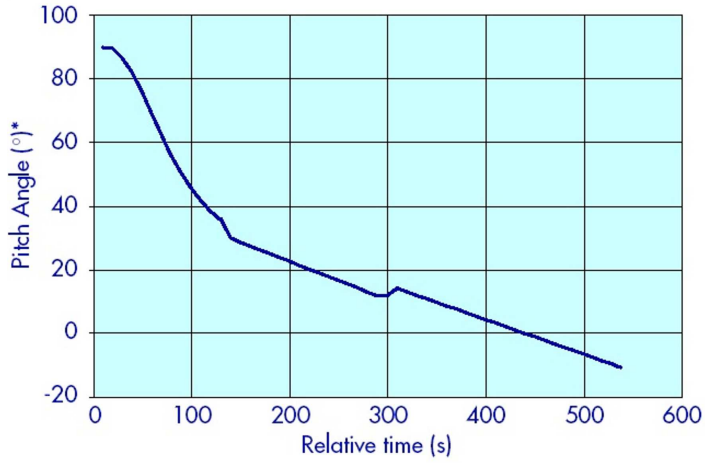

Figure 7.52: ***Reference pitch law for a Soyuz launch. The Galilean Pitch Angle is shown as a function of time. The pitch law shown in Figure 7.51 was derived from this plot. (Taken from [15])***

First stage	
Total thrust	4· 1021000 N
Angle between thrust and spacecraft main axis	0°
Specific impulse	3129.39 m/s
Total propellant mass	156640 kg
Separation mass	19874.3 kg
Burn time	0-118.3 sec
Separation time	118.3 sec
Second stage	
Total thrust	990200 N
Angle between thrust and spacecraft main axis	0°
Specific impulse	3129.39 m/s
Total propellant mass	90100 kg
Separation mass	8165 kg
Burn time	0-278 sec
Separation time	278 sec
Third stage	
Total thrust	298000 N
Angle between thrust and spacecraft main axis	0°
Specific impulse	3521.79 m/s
Total propellant mass	25400 kg
Separation mass	2026 kg
Burn time	280-578 sec
Separation time	578 sec

Table 7.9: ***Characteristics of the three stage Soyuz launcher as used for the trajectory computation.***

characteristics of all stages as assumed for the computation. If one compares the properties of the Soyuz launcher with those of the Ariane 44LP it shows that the Russian rocket is much more powerful. That indicates that the Soyuz rocket reaches the desired injection parameters much earlier. Hence, the flight profile looks completely different compared to that of an Ariane launcher.

For the computation of the Soyuz trajectory three additional mass items are considered. The fairing with a mass of 2700 kg is jettisoned after 203 seconds. Furthermore, inert mass is ejected at a constant rate during the firing of the first and second stage. It is assumed that during the first 118.3 seconds 3500 kg are ejected and, in addition to that, during the first 278 seconds another 700 kg. The liftoff takes place instantaneously at $t = 0$ seconds. For the reference surface a value of $S = 23$ m^2 is chosen. In Table 7.10 the drag coefficient C_D is given as a function of the Mach number. The lift coefficient C_L is set to zero. The gravitational potential of the Earth is assumed to be spherically symmetric. For the atmosphere the analytical model is selected.

Mach Number	**Drag coefficient** C_D	**Mach Number**	**Drag coefficient** C_D
<0.7	2.22	1.1	3.08
0.8	2.17	1.5	2.4
0.9	2.46	2.0	1.92
1.0	3.09	>3.0	1.28
1.05	3.2		

Table 7.10: ***The drag coefficient*** C_D ***of the Soyuz launcher as a function of the Mach number. (Courtesy of ESA/EADS)***

While the launch of the Ariane 44LP of the previous example takes place in the equatorial plane, this configuration shall present a more realistic trajectory. Thus, the position of the launch pad and the initial azimuth of the relative velocity have to be specified. The launch pad is chosen to be one of the European spaceport in Kourou in French Guiana. Its geographic coordinates are $l_0 = 307.22^\circ\, E$ and $L_0 = 5.24^\circ\, N$. The launch azimuth needs to be set to $\chi_0 = 40.22^\circ$ in order to reach a transfer orbit leading finally to the desired orbit of the ISS. The integration time step is set to 0.1 seconds like in the previous example.

Figures 7.53, 7.55, 7.56 and 7.58 show the results of the trajectory computation. Corresponding plots from *[15]* are presented in parallel in Figures 7.54 and 7.57. Like in the previous section it shows that the general trajectory properties are reproduced very well. An exact copy of the reference plots, however, is not achieved as uncertainties in the pitch law and the mass and thrust history can not be avoided.

As expected a comparison between the launch trajectory of the Soyuz rocket and that of the Ariane 44LP launcher reveals completely different flight profiles (see Figures 7.53 and 7.43). This results mainly from different booster configurations and the differences in the imposed pitch law enabling the Soyuz rocket to reach its desired altitude directly and much faster. The influence of the pitch law on the trajectory is also investigated in the next section where a different pitch law is imposed on the Soyuz launcher configuration already used in this section.

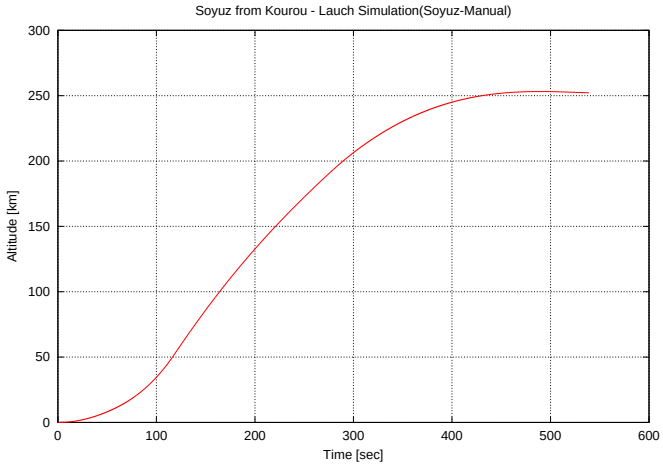

Figure 7.53: ***Soyuz Launch Trajectory (1): Altitude vs. Time***

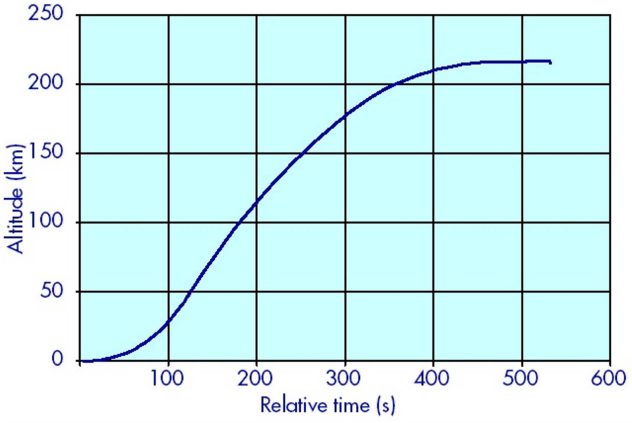

Figure 7.54: ***Reference trajectory for a typical Soyuz launch showing the altitude as a function of time. (Taken from [15] for comparison)***

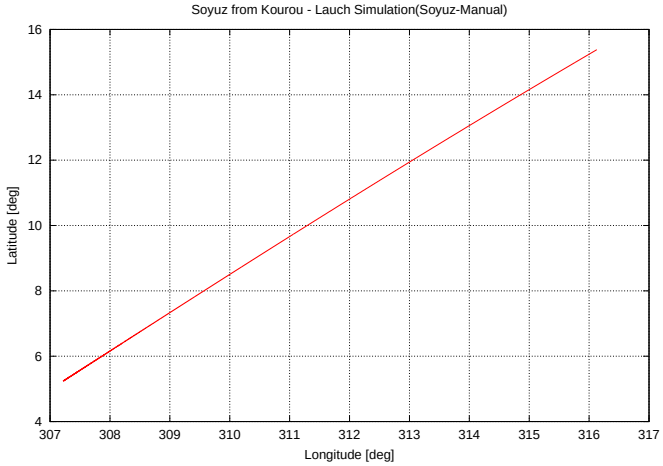

Figure 7.55: ***Soyuz Launch Trajectory (1): Latitude vs. Longitude. The ground-track shows that the rocket is on the way to an inclined orbit.***

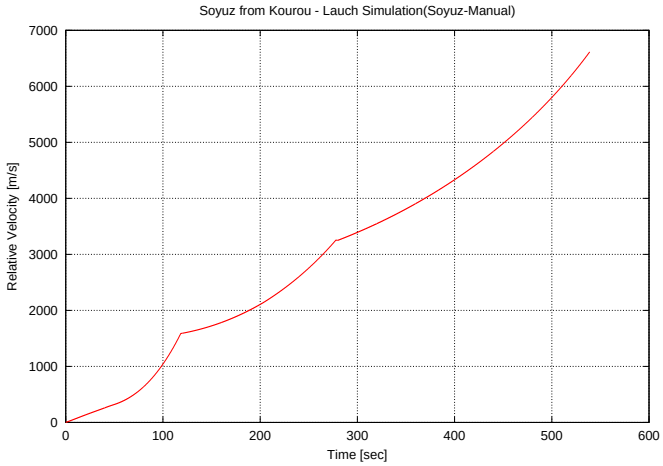

Figure 7.56: ***Soyuz Launch Trajectory (1): Relative Velocity vs. Time***

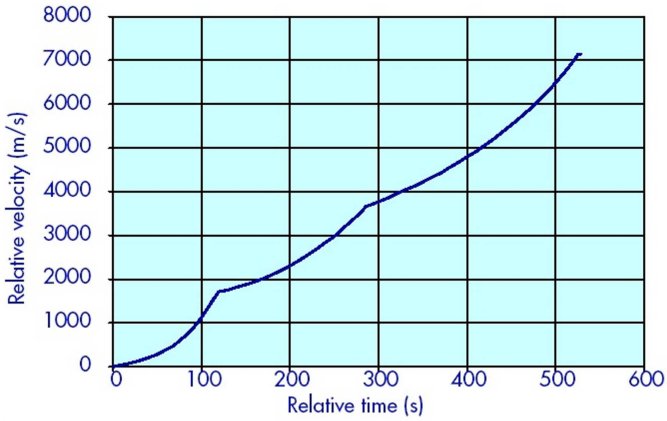

Figure 7.57: ***Reference trajectory for a typical Soyuz launch showing the relative velocity as a function of time. (Taken from [15] for comparison)***

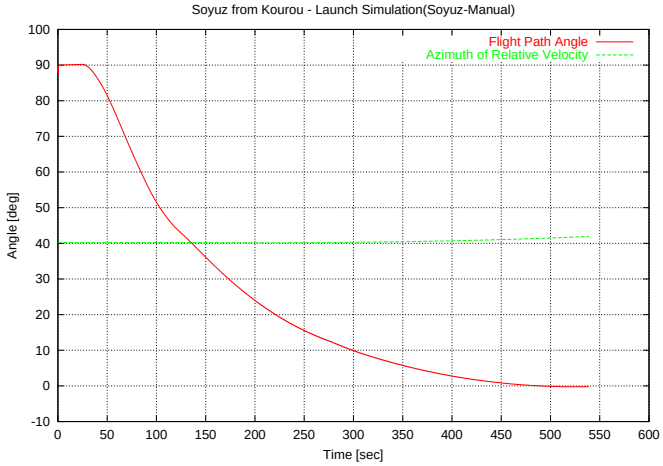

Figure 7.58: ***Soyuz Launch Trajectory (1): Flight Path Angle and Azimuth of the Relative Velocity vs. Time***

7.3.3 Soyuz: Launch from Kourou (2)

As mentioned at the beginning of the previous section the following example is very similar to the one presented there. However, there is one important difference: The imposed pitch law. Instead of using the values of Table 7.8, a second pitch law is used for which Table 7.11 contains the approximated analytical expressions. This pitch law is also plotted in Figure 7.59 and is derived from Figure 7.60.

Time period [s]	ϑ_I **at the beginning**	ϑ_I **at the end**
$0.0 < t < 12.0$	$90.0°$	$90.0°$
$12.0 < t < 17.9$	$90.0°$	$86.3°$
$17.9 < t < 20.0$	$86.3°$	$86.3°$
$20.0 < t < 36.8$	$86.3°$	$76.8°$
$36.8 < t < 61.3$	$76.8°$	$60.0°$
$61.3 < t < 80.6$	$60.0°$	$46.3°$
$80.6 < t < 100$	$46.3°$	$36.7°$
$100.0 < t < 117.0$	$36.7°$	$30.2°$
$117.0 < t < 120.0$	$30.2°$	$35.8°$
$120.0 < t < 205.0$	$35.8°$	$25.2°$
$205.0 < t < 319.0$	$25.2°$	$12.5°$
$319.0 < t < 400.0$	$12.6°$	$4.4°$
$400.0 < t < 450.0$	$4.4°$	$-1.0°$
$450.0 < t < 567.0$	$-1.0°$	$-12.6°$

Table 7.11: ***Pitch law for the computation of a Soyuz launch from Kourou as derived from Figure 7.60 provided by ESA/EADS [20]. The values refer to a Galilean reference frame, i.e. they are measured with respect to the launch pad.***

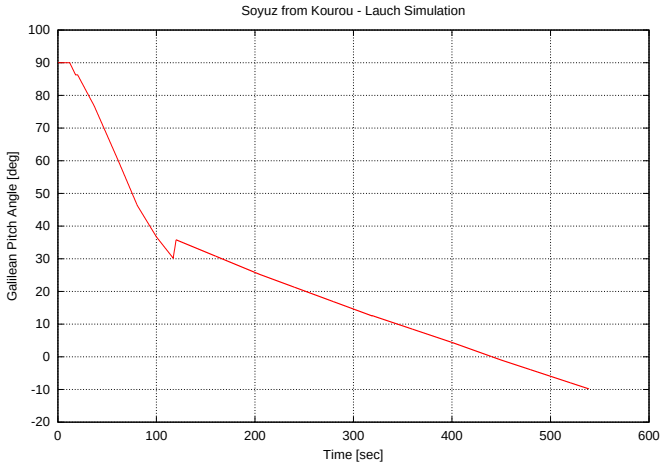

Figure 7.59: ***Soyuz Launch Trajectory (2): Galilean Pitch Angle vs. Time. This pitch law is summarized in Table 7.11.***

Like in the previous section the launch of a Soyuz rocket from Kourou is computed. Hence, the input parameters equal exactly those of the previous section: The evolution of the thrust

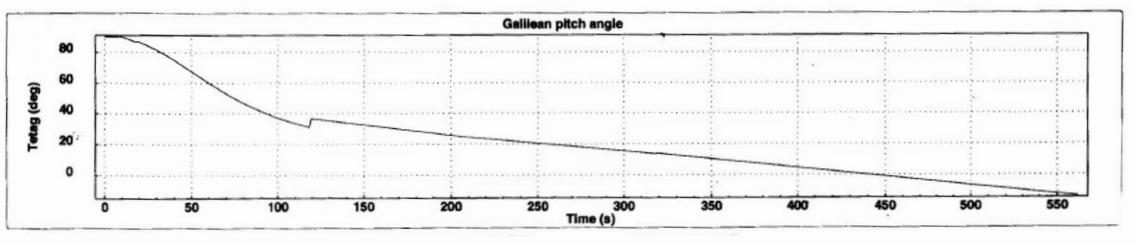

Figure 7.60: ***Reference pitch law for a Soyuz launch from Kourou. The Galilean Pitch Angle is shown as a function of time. This plot served as a basis for the derivation of the pitch law shown in Figure 7.59. (Courtesy of ESA/EADS [20])***

and the mass, prescribed by Table 7.9, as well as the drag coefficient (Table 7.10) and the launch pad coordinates are identical. The integration time step is set to 0.1 seconds. For the atmosphere the analytical model is selected and the gravitational field is assumed to be spherically symmetric.

The computed results are shown in Figures 7.61, 7.63, 7.65, 7.67 and 7.68. Corresponding figures from ESA/EADS are shown in parallel. While the results match very well the reference plots, one will notice that they look different compared to those of the previous section. The altitude after 500 seconds, for instance, is now almost 50 km lower than in the previous example (Figure 7.61), whereas the relative velocity is slightly higher (Figure 7.63). A comparison of Figure 7.68 with Figure 7.55, both showing the ground-track of the trajectory, reveals also differences. They all result from the variations in the imposed pitch law underlining its important role in leading a rocket to the desired injection parameters.

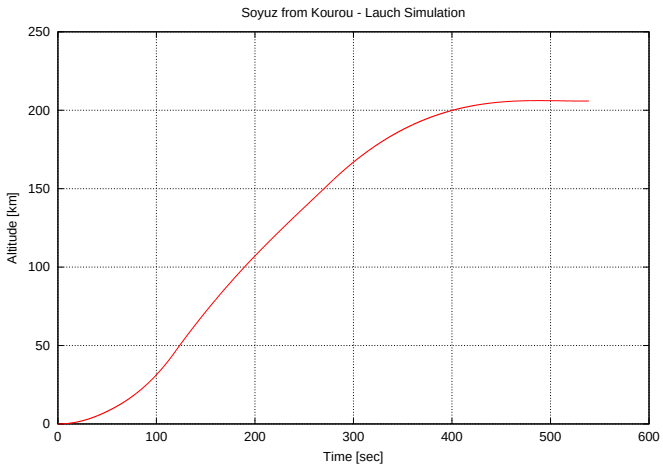

Figure 7.61: ***Soyuz Launch Trajectory (2): Altitude vs. Time. Compared to Figure 7.53 the altitude after 500 seconds is almost 50 km lower.***

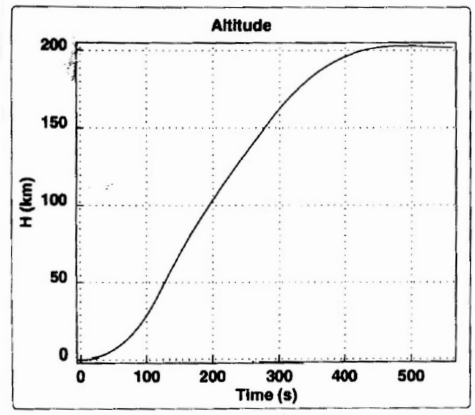

Figure 7.62: ***Reference trajectory for a Soyuz launch from Kourou. The altitude is shown as a function of time. (Courtesy of ESA/EADS [20])***

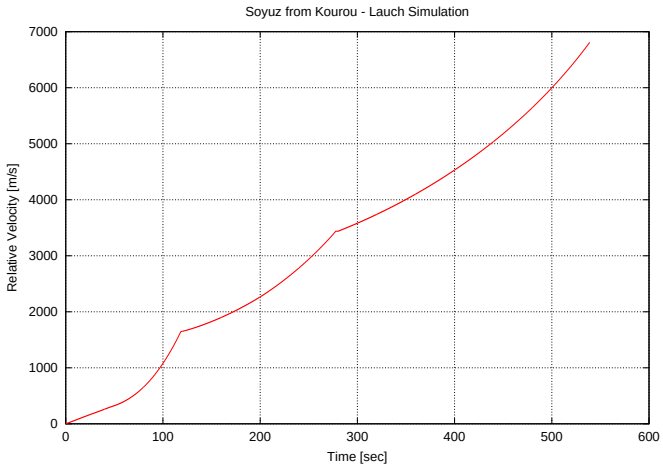

Figure 7.63: ***Soyuz Launch Trajectory (2): Relative Velocity vs. Time. Compared to Figure 7.56 the velocity in this plot is slightly higher.***

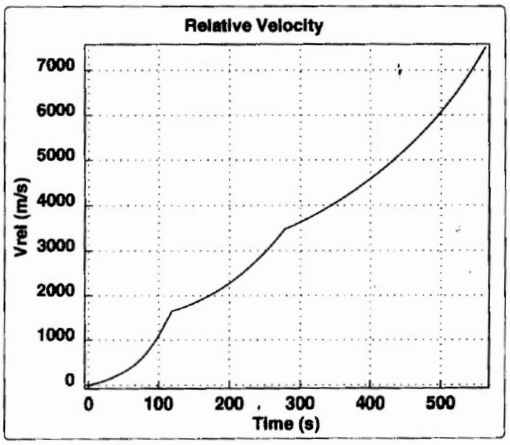

Figure 7.64: ***Reference trajectory for a Soyuz launch from Kourou. The relative velocity is shown as a function of time. (Courtesy of ESA/EADS [20])***

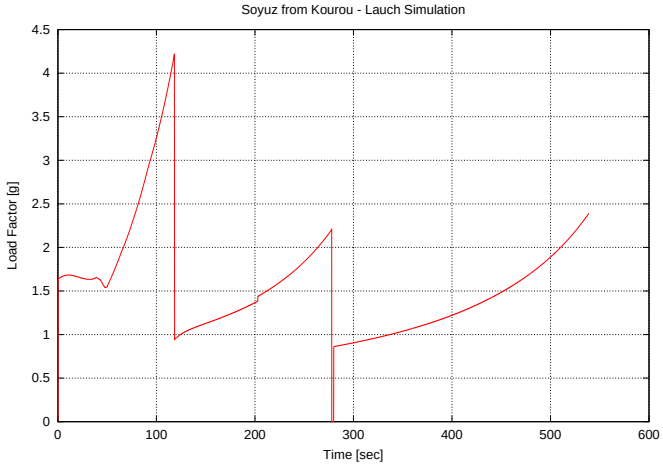

Figure 7.65: ***Soyuz Launch Trajectory (2): Load Factor vs. Time***

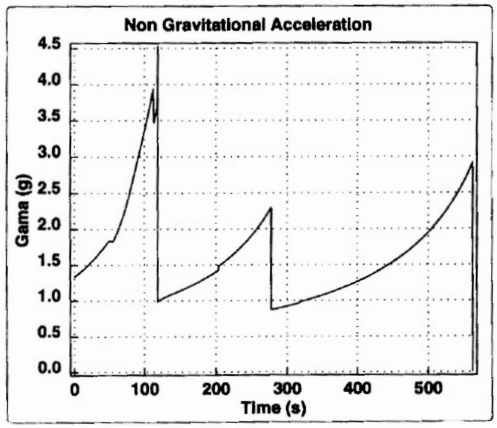

Figure 7.66: ***Reference trajectory for a Soyuz launch from Kourou. The load factor is shown as a function of time. (Courtesy of ESA/EADS [20])***

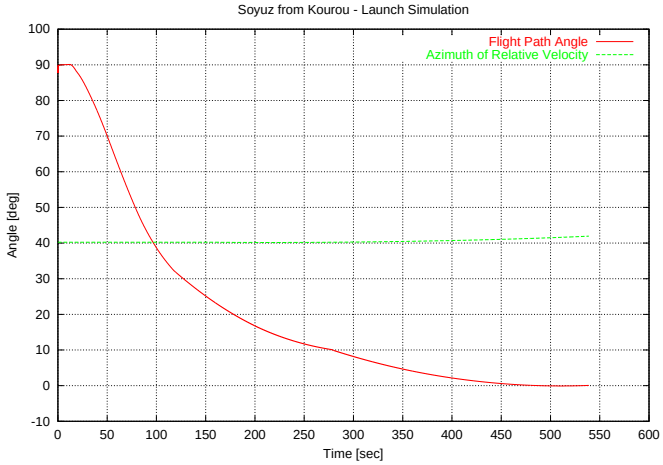

Figure 7.67: ***Soyuz Launch Trajectory (2): Flight Path Angle and Azimuth of the Relative Velocity vs. Time***

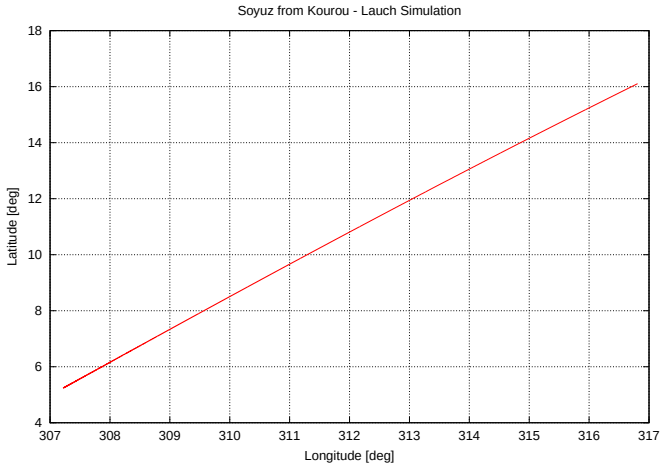

Figure 7.68: ***Soyuz Launch Trajectory (2): Latitude vs. Longitude. It can be seen that the rocket is on its way to an inclined orbit. Differences between this plot and Figure 7.55 arise due to differences in the imposed pitch laws.***

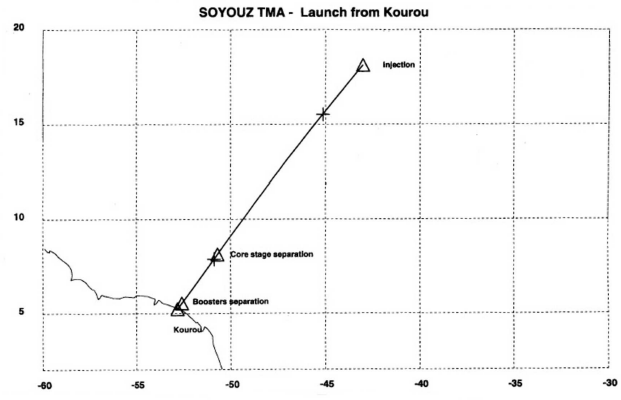

Figure 7.69: ***Reference trajectory for a Soyuz launch from Kourou. The latitude is shown as a function of the longitude. (Courtesy of ESA/EADS [20])***

Figure 7.70: ***A Soyuz rocket lifting off. (Courtesy of ESA)***

Chapter 8

The Quality of the Numerical Integrator

In this chapter the long term behavior of the numerical integrator scheme is analyzed in greater detail. This offers the possibility to compare the performance of the 8^{th}-order Runge-Kutta algorithm to that of other numerical integrators commonly used for astrophysical computations. Comparing the different schemes is, however, not as easy as it might appear. This arises due to the fact that in most accessible scientific literature the quality of integrators for orbital mechanics is deduced from errors in the eccentricity or the semi-major axis of the orbit as the computations are normally based on the six classical orbital elements (Figure 2.1). Since in the mathematical approach presented here neither the eccentricity nor the semi-major axis is part of the equations, a direct comparison of the numerical schemes is not possible. Therefore the following actions are taken in order to analyze the numerical scheme and to make a (limited) comparison possible:

1. **Derivation and analysis of the eccentricity from the computed results:** By comparing numerical results for the eccentricity of different orbits with analytical results, the quality of the former ones can be derived. Although the eccentricity of the computed classical Keplerian orbits is not an output variable and was of minor interest (see subsections 7.1.1.-7.1.3.), its value can be obtained from the results of the numerical integration by different means. Here, the focus is on the following three methods:

   - *Three Point Method:* Three points of the orbit (equally distributed along the trajectory) are taken and the shape of the ellipse is calculated by using the basic equation of an ellipse.
   - *Least Squares Method:* A certain number of points along the trajectory is selected and the eccentricity is computed by using a least squares fit.
   - *Runge-Lenz-Vector:* The Runge-Lenz-Vector is calculated as its absolute value equals the eccentricity.

2. **Derivation and analysis of the apogee and perigee from the computed results:** Following the approach presented above the apogee and the perigee of the orbit are calculated. Again, the least squares method is applied and the long term behavior of these variables is investigated.

3. **Analysis of the behavior of the radius vector r:** This variable is part of the differential equations (3.21). By comparing its computed value to another one obtained from the solution of an analytical expression for r the accuracy of the integrator scheme can be derived.

4. **Calculation and analysis of the angular momentum:** Since the angular momentum should be conserved during an unperturbed orbit propagation, changes in its value are an indicator for the quality of the numerical scheme.

5. **Analysis of the behavior of the latitude L and the azimuth χ:** For an equatorial orbit these variables should remain constant. Thus, for such an orbit any changes in these variables are based on numerical errors produced by the integrator.

8.1 The performance of the Runge-Kutta algorithm

All methods described above are investigated. Different configurations and conditions are selected (e.g. different eccentricities) in order to get a better understanding of the behavior of the numerical scheme. The obtained results are presented in the following subsections.

8.1.1 Errors in the eccentricity

As mentioned above three different methods are used to calculate the eccentricity of the orbit. Each approach is discussed in detail.

Three Point Method:

By taking three points of each orbit the eccentricity of the orbital ellipse can be calculated, as shown in the following.

An ellipse is completely defined by

$$\frac{(x-c)^2}{a^2} + \frac{y^2}{b^2} = 1 \tag{8.1}$$

where a is the semi-major axis, b the semi-minor axis and c a shift required to put one focus of the ellipse in the origin of the x-y-plane. The eccentricity e can be calculated via the relation

$$e = \frac{c}{a} \quad . \tag{8.2}$$

Let the ellipse now represent a satellite orbit. Then x and y can be computed easily if the Earth is assumed to be non-rotating ($\vec{\Omega} = 0$). In this case it is

$$x = -r(t) \cdot \cos l(t), \qquad y = -r(t) \cdot \sin l(t) \tag{8.3}$$

with $l(t)$ as geographic longitude and Greenwich pointing in negative x direction, and $r(t)$ being the altitude vector.

Thus, since x and y are known and prescribed, equation (8.1) has to be solved for a, b and c. Define

$$\alpha := \frac{1}{a^2}, \quad \beta := -\frac{2c}{a^2}, \quad \gamma := \frac{1}{b^2} \tag{8.4}$$

and get

$$\alpha x^2 + \beta x + \gamma y^2 = 1 - \frac{\beta^2}{4\alpha} \quad . \tag{8.5}$$

By taking three values for x and y the following system of equations is obtained

$$\begin{pmatrix} x_1^2 & x_1 & y_1^2 \\ x_2^2 & x_2 & y_2^2 \\ x_3^2 & x_3 & y_3^2 \end{pmatrix} \begin{pmatrix} \alpha \\ \beta \\ \gamma \end{pmatrix} = \left(1 - \frac{\beta^2}{4\alpha}\right) \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix} \quad .$$

It is possible to solve this system explicitly for α, β and γ (e.g. by applying Cramer's Rule) from which the required values of a, b and c can be derived via (8.4). The eccentricity e follows then with equation (8.2).

The results of the Three Point Method for two orbits with different eccentricities ($e = 0.58$, $e = 0.27$) are shown in Figure 8.1. In both cases each orbit consists of 63 integration steps. It

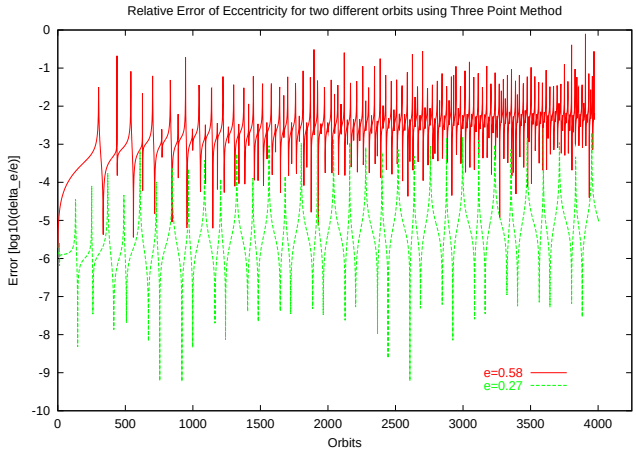

Figure 8.1: ***Errors in the eccentricity for two different orbit configurations. In each case three out of 63 points per orbit are taken in order to compute the eccentricity via the Three Point Method. The theoretical values for the eccentricities are e=0.57 and e=0.27.***

shows that the error for the more eccentric orbit is remarkably bigger. However, in both cases the error varies over several orders of magnitude in relative short time periods and reaches a maximum of approximately 10 % for the more eccentric orbit. Although one can already see, that fitting these curves might lead to better and more plausible results, the behavior of the Three Point Method is to be examined in greater detail, in order to check whether the fluctuations are produced by integrator or whether they result from the applied method.

By analyzing the Three Point Method more accurately it shows that numerical effects can overlap the errors produced by the integrator. The implementation of the Three Point Method is very sensitive with respect to the integration time step and hence also with respect to the selected three points and their position along the orbit. Figure 8.2 shows the results for two computations of an orbit with $e = 0.27$, where the only difference is a small modification of the integration time step. One curve is very similar to the results shown in Figure 8.1: The error varies over a wide range of magnitudes. The second curve, however, is very smooth,

and the fluctuations have disappeared (except for one minimum). Nevertheless, fitting both curves would lead to comparable results. Thus, one can conclude that numerical effects are the reason for the big fluctuations in Figures 8.1 and 8.2 and that the results obtained with the implemented Three Point Method are very sensitive with respect to their input.

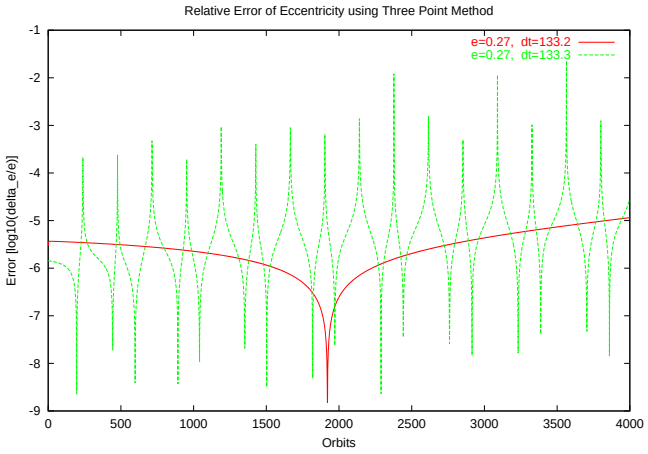

Figure 8.2: ***Errors in the eccentricity for two different integration time steps when the Three Point Method is applied. The orbit has an eccentricity of e=0.27 and consists of approximately 63 integration steps. Thus, the results are comparable to those in Figure 8.1. Although the orbit configuration is exactly the same and the integration time step differs only by 0.1 seconds, the two plots look completely different, possibly due to numerical effects.***

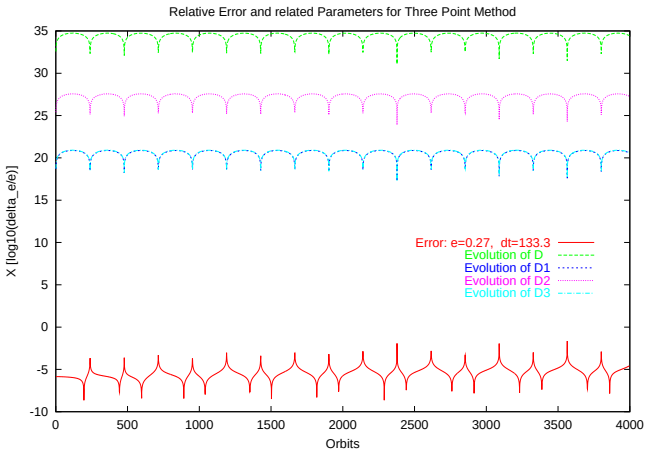

Figure 8.3: ***Errors in the eccentricity for an orbit with e=0.27 and underlying parameters for the Three Point Method. The curve labelled "Error" corresponds to the fluctuating plot already depicted in Figure 8.2. Additionally, the evolution of the values of four determinants (D-D3) used for the computation of the eccentricity is shown. It seems as if there is a concrete connection between the fluctuations in the error and the corresponding values of the determinants.***

Apparently, some combinations of the three points, that are selected along the orbit, lead to singularities and cause fluctuations when the eccentricity is computed. The current implementation requires the computation of several determinants when the system of equation shown above is to be solved. Here, a possible reason for the occurring singularities might be hidden, as the computation of these determinants can be problematic when the values are very small or very big. Figure 8.3 shows again one of the error curves already depicted in Figure 8.2. In parallel the evolution of different determinants used for the computation of e is presented. As one can see, the different maxima in the error are directly connected to the minima in the values for the determinants. This underlines that most likely the fluctuations are caused by singularities occurring during the computation of the eccentricity[1].

In order to avoid the singularities and consequently also the fluctuations, it seems reasonable to validate the results by applying the Least Squares Method to the data. This methods considers all 63 values per orbit for x and y instead of taking only three points to calculate the eccentricity. Thus, problematic combinations of three points are no longer possible.

Least Squares Method:

This method is also based on equation (8.1) like the Three Point Method. Rewriting this equation as

$$x^2 - 2cx + \frac{a^2}{b^2}y^2 = a^2 - c^2 \tag{8.6}$$

and defining

$$\alpha := a^2 - c^2, \quad \beta := \frac{a^2}{b^2} \tag{8.7}$$

leads to

$$x^2 - 2xc + \beta y^2 = \alpha \quad . \tag{8.8}$$

Since a certain number of values for x and y is given the best least squares fit for the unknown variables α, β and c is found where it holds

$$\langle (x^2 - 2cx + \beta y^2 - \alpha)^2 \rangle \quad \rightarrow \quad Min \quad . \tag{8.9}$$

Thus, one demands

$$\frac{\partial}{\partial \alpha}\langle (x^2 - 2cx + \beta y^2 - \alpha)^2 \rangle = 0 \quad ! \tag{8.10}$$

$$\frac{\partial}{\partial \beta}\langle (x^2 - 2cx + \beta y^2 - \alpha)^2 \rangle = 0 \quad ! \tag{8.11}$$

$$\frac{\partial}{\partial c}\langle (x^2 - 2cx + \beta y^2 - \alpha)^2 \rangle = 0 \quad ! \tag{8.12}$$

Respecting the partial derivations, these equations can be transformed into the following system

$$\begin{pmatrix} 1 & -\langle y^2 \rangle & 2\langle x \rangle \\ \dfrac{\langle y^2 \rangle}{\langle y^4 \rangle} & -1 & 2\dfrac{\langle xy^2 \rangle}{\langle y^4 \rangle} \\ \dfrac{\langle x \rangle}{2\langle x^2 \rangle} & -\dfrac{\langle y^2 x \rangle}{2\langle x^2 \rangle} & 1 \end{pmatrix} \begin{pmatrix} \alpha \\ \beta \\ c \end{pmatrix} = \begin{pmatrix} \langle x^2 \rangle \\ \dfrac{\langle x^2 y^2 \rangle}{\langle y^4 \rangle} \\ \dfrac{\langle x^3 \rangle}{2\langle x^2 \rangle} \end{pmatrix}$$

[1] An enhanced implementation of the Three Point Method, yet to be investigated, might be more stable.

Like in the previous section Cramer's Rule can be used to solve this system for α, β and c. From that, a and b can be obtained via (8.7) and the eccentricity e follows directly via (8.2).

Figure 8.4 shows the results obtained with the Least Squares Method. By comparing these results to those of the Three Point Method (Figure 8.1) it shows that the variations in the errors have disappeared while the different orders of magnitude of the error for the different eccentricities are still present.

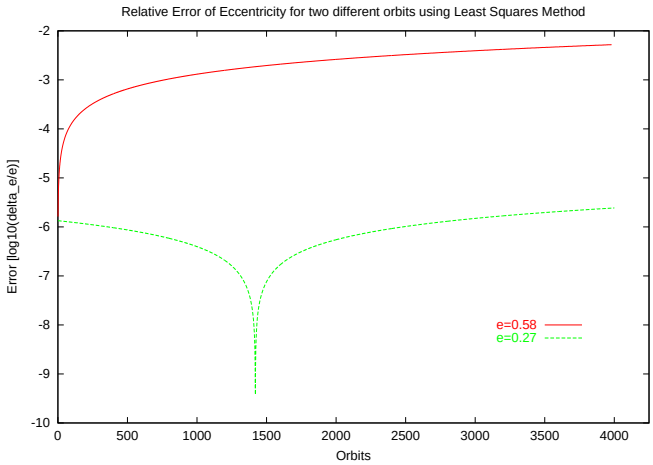

Figure 8.4: ***Errors in the eccentricity for two different orbit configurations. In each case 63 points per orbit are taken and the Least Squares Method is applied. The theoretical values for the eccentricity are e=0.58 and e=0.27.***

The minimum appearing in Figure 8.4 in the plot for the orbit with $e = 0.27$ results from the fact that the computed orbit undergoes the following evolution: At the beginning the eccentricity is underestimated compared to the theoretical value. Then the eccentricity begins to increase steadily and in consequence it reaches approximately the theoretical value at some point in time (the minimum of the error). Finally the orbit gets more and more eccentric and the error increases again.

Runge-Lenz-Vector:

The Runge-Lenz-Vector is defined as

$$\vec{e} = \frac{\vec{V} \times \vec{M}_r}{\gamma_G(m_1 + m_2)} - \frac{\vec{r}}{r} \quad , \tag{8.13}$$

where $\vec{V}$ is the velocity, $\vec{M}_r = \vec{r} \times \vec{V}$ represents the reduced angular momentum, γ_G is the gravitational constant and m_1 and m_2 are the masses of the bodies orbiting each other. Since $|\vec{e}|$ equals the eccentricity of the orbit this method offers another possibility to compute the error in the eccentricity. By applying basic geometry $|\vec{e}|$ can be computed from the variables r, V and γ. Figure 8.5 shows that the results obtained with the Runge-Lenz-Vector fully confirm those of the previous methods.

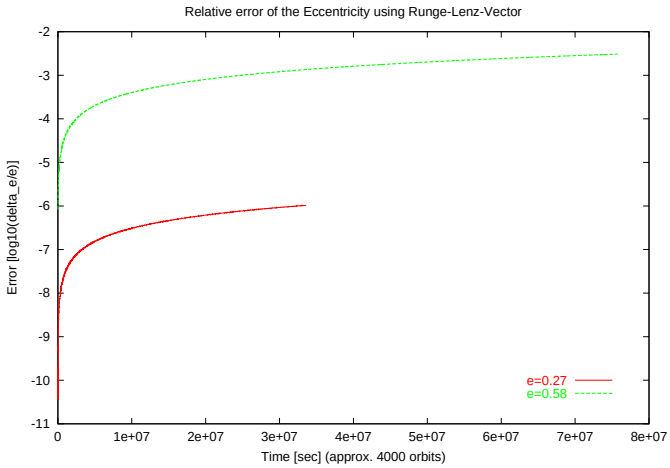

Figure 8.5: ***Errors in the eccentricity for two different orbit configurations. The errors are calculated from the absolute value of the Runge-Lenz-Vector. The theoretical values for the eccentricity are e=0.58 and e=0.27.***

8.1.2 Errors in the apogee and perigee

From equation (8.6) and the results following thereafter the apogee and the perigee of the orbit can be calculated via

$$r_{apo} = a + c, \qquad r_{peri} = a - c \tag{8.14}$$

after the Least Squares Method has been applied. Thus, not only the long term behavior of the eccentricity but also that of the apogee and perigee can easily be analyzed.

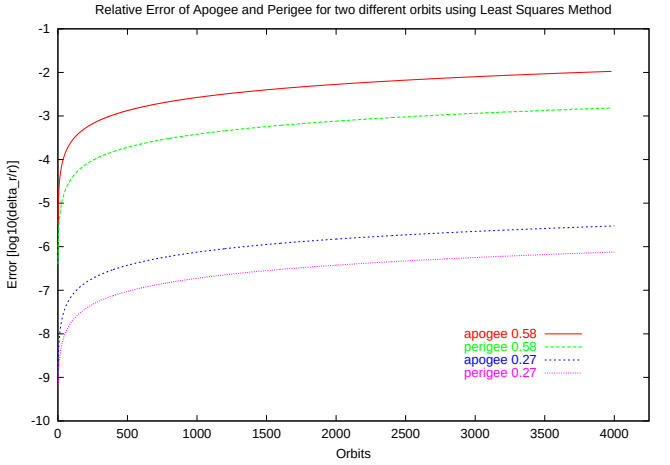

Figure 8.6: ***Errors in the apogee and perigee for two different orbit configurations. In each case 63 points per orbit are taken and the Least Squares Method is applied. The theoretical values for the eccentricity are e=0.58 and e=0.27.***

The results of this computation are plotted in Figure 8.6. It shows that the errors in the apogee are much bigger than those in the perigee. Furthermore, the errors for the highly eccentric orbit are again much bigger than those for the orbit with a smaller eccentricity. This behavior is known for other kinds of astrophysical many body integrators, too (see *[27]*).

8.1.3 Errors in the radius vector

By analyzing the behavior of the radius vector r, a variable is considered that is used directly in the differential equations. The results of the numerical integrator can be compared to those obtained from the equation of a conic section in polar coordinates

$$r = \frac{p}{1 + e \cdot \cos(\theta)} \tag{8.15}$$

with $p = (a(1 - e^2))$ being the focal parameter. e and a are prescribed and if a non-rotating Earth is assumed, θ equals the longitude l used in differential equations. Thus, if the actual value of l is plugged into (8.15) the corresponding value for r is obtained for each time step. The result can then be compared to the actual value calculated by the integrator.

For the computation 63 integration steps per orbit are taken. Figure 8.7 and Figure 8.8 show the results. They are in good agreement with those of the previous sections. For a circular orbit, however, the errors are remarkably smaller and corresponds to those of the latitude for an equatorial orbit shown in 8.1.5 later in this chapter.

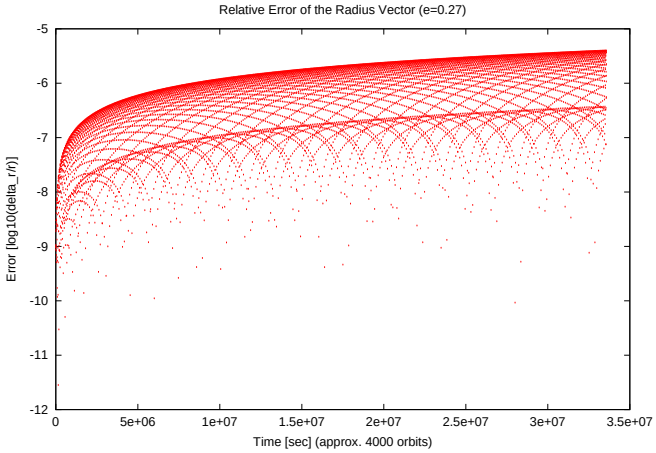

Figure 8.7: ***Errors in the radius vector for an orbit with an eccentricity of 0.27***

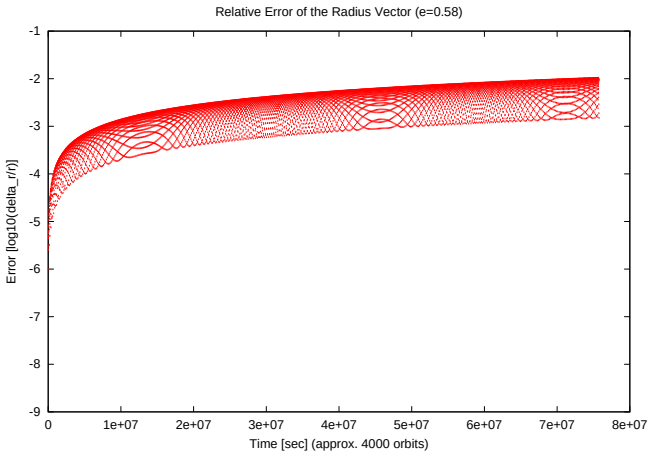

Figure 8.8: ***Errors in the radius vector for an orbit with an eccentricity of 0.58***

8.1.4 Errors in the angular momentum

If there are no additional perturbations to the orbit and no interactions with other bodies the absolute value of the angular momentum $\vec{M}$ of the elliptical motion shall be conserved. In order to check to which extent this holds true, the angular momentum has to be computed from the given variables. A general expression for $\vec{M}$ is

$$\vec{M} = m \cdot (\vec{r} \times \vec{V}) \quad . \tag{8.16}$$

Accordingly, for the rotating Earth reference frame, the reduced angular momentum can be written as

$$M_r = r \cdot V \cos(\gamma) \quad , \tag{8.17}$$

where the mass of the spacecraft m is neglected. γ represents the flight path angle, r and V are the radius vector and the relative velocity. Via (8.17) the actual value of the angular momentum can be calculated for every integration step.

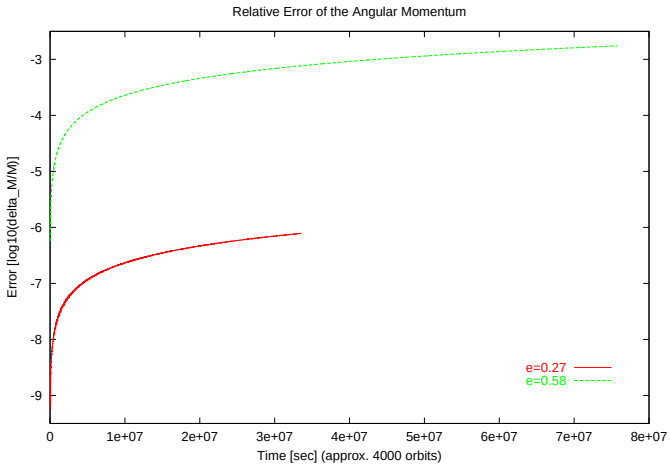

Figure 8.9: ***Errors in the angular momentum for two different orbit configurations. The theoretical values of the eccentricity are e=0.58 and e=0.27.***

The evolution of the errors in the angular momentum is shown in Figure 8.9. Again, the results underline the fact that the error for orbits with a higher eccentricity is bigger than that for less eccentric orbits. In both cases the errors have the same order of magnitude like the errors in the radius vector of the previous section.

8.1.5 Errors in the latitude and azimuth

Finally, the accuracy of the numerical scheme is investigated by analyzing two variables of the differential equations that should remain constant for an equatorial orbit: The azimuth of the relative velocity χ and the latitude L. Whatever the eccentricity might be, for an equatorial orbit the azimuth should remain $\chi = 90°$ and the latitude should be constantly $L = 0°$.

For 63 integration steps per orbit and two different eccentricities the results of the computation are shown in Figure 8.10 and Figure 8.11. The order of magnitude of the errors is in both cases significantly lower than in the plots shown previously in this chapter. Furthermore, at least for the orbit with a lower eccentricity, the errors seem to remain more or less constant,

while in all other cases the errors increased steadily with time. It shows again, that the results for a less eccentric orbit are more accurate.

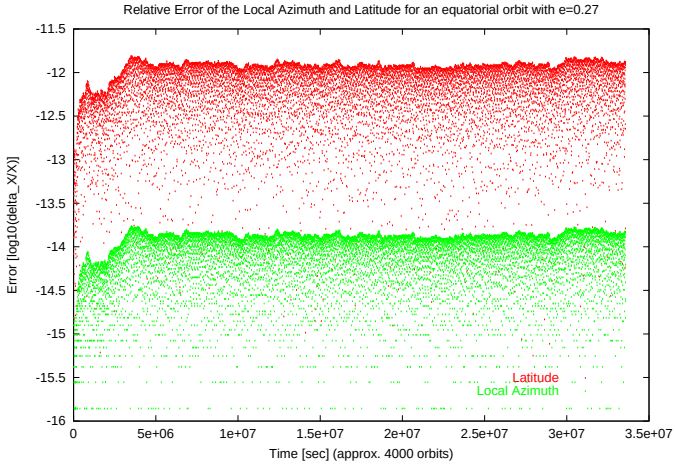

Figure 8.10: ***Errors in the latitude and the azimuth for an equatorial orbit with an eccentricity of e=0.27. The errors are much smaller than those of the previous subsections and they do not seem to increase constantly in time.***

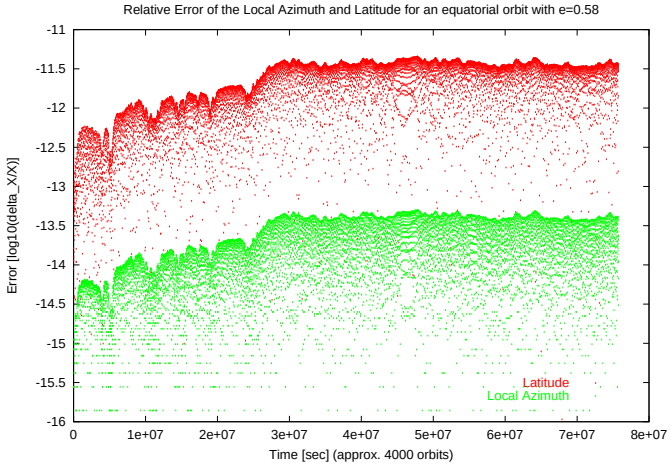

Figure 8.11: ***Errors in the latitude and azimuth for an equatorial orbit with an eccentricity of e=0.58. The errors are much smaller than those of the previous subsections. However, they are slightly bigger than for the less eccentric orbit shown above.***

8.1.6 Summary of the results

The previous sections show, that the results for the performance of the 8^{th}-order Runge-Kutta algorithm, obtained with the different methods, are in most cases very similar. However, a short summary is to be given here, as the results are compared to those of other numerical integration schemes in the following.

1.) The orbital plane is very well conserved. Section 8.1.5 shows that for an equatorial orbit the errors for L and χ range from 10^{-11} to 10^{-14} depending on the eccentricity.

2.) The dependency on the eccentricity is confirmed by the errors of the other analyzed variables (e.g. radius vector, angular momentum, eccentricity). For these variables the errors range from $10^{-2}-10^{-3}$ to $10^{-5}-10^{-6}$ depending on the orbit configuration. A remarkable shift in the perigee is, however, not observed.

3.) All examples demonstrate that the errors are not oscillating but constantly increasing in time[2]. Exceptions are the results of the Three Point Method (discussed below in 4.) and those for L and χ, where the error seems to remain constant (see 8.1.5).

4.) The use of the here presented Three Point Method for the computation of the eccentricity is problematic, as numerical effects can lead to huge fluctuations in the error for certain integration time steps and for certain combinations of the three selected points. This dependency on the input makes this method impracticable. However, it seems plausible that a different implementation of the Three Point Method, avoiding the occurring singularities, leads to more reliable results.

After having analyzed and discussed the 8^{th}-order Runge-Kutta algorithm in detail a comparison to the performance of other integration schemes is provided in the following.

8.2 The performance of other numerical integrators and comparison

It was pointed out previously that a comparison between the results of the 8^{th}-order Runge-Kutta algorithm and other numerical integrators is not very easy. For example, it would be inadequate to compare the results for the eccentricity directly if the other integrators were applied to the classical formulation of the differential equations, namely the classical orbital elements (Figure 2.1 and equations (2.1)-(2.6)). A real comparison would only be possible if different schemes were implemented solving the same set of equations. This is, however, not the main objective of this work. Thus, in the following some examples describing the performance of other mathematical schemes are presented, which give an overview of the typical accuracy of numerical integrators used in astrophysical computations.

Figures 8.12, 8.13 and 8.14 are taken from *[17]* and show errors in the computation of perturbed binary systems for different numerical schemes. The errors refer either to the eccentricity of the orbit or to the computed energy of the perturbed system. The perturbations are caused by a distant galaxy and can be described by a tidal potential

$$U_{tid} = C(-2x^2 + y^2 + z^2) \quad . \tag{8.18}$$

By changing the constant C in (8.18), different strengths of the perturbation can be studied. The potential U_{tid} causes variations in the eccentricity of the binary system, so that for a maximum relative perturbation of 0.006 (Figures 8.12 and 8.14) one finds a minimum of $e = 0.258$ and a maximum $e = 1$. In case of a stronger relative perturbation of 0.2 (Figure 8.13) the minimum of the eccentricity is slightly smaller while the maximum is still near $e = 1$ (see *[17]*).

[2]While increasing, the error of the radius vector varies over a restricted range of magnitudes (see 8.1.3).

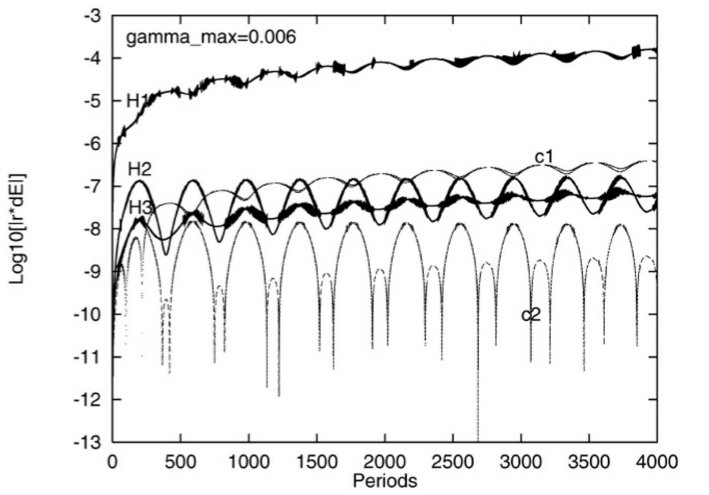

Figure 8.12: ***Errors in the energy in a moderately perturbed binary system for different mathematical schemes. The plot was taken from [17] as an example of the accuracy of other numerical integrators. The plots labelled H1, H2 and H3 give errors for Hermite schemes whereas the curves labelled with c1 and c2 plot the errors for a method using Stumpff's c-functions.***

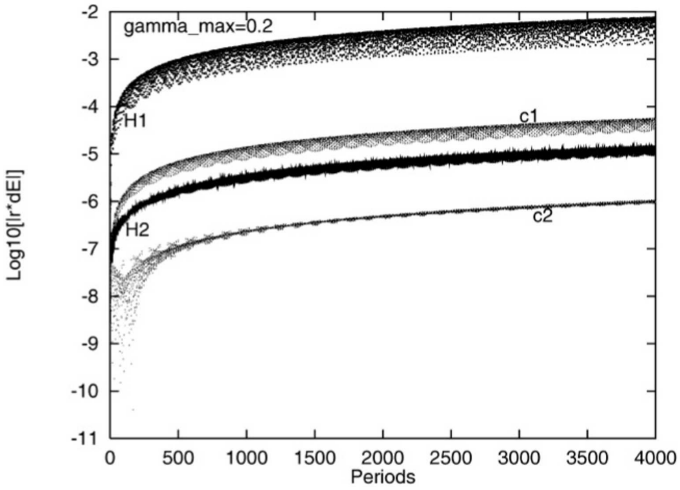

Figure 8.13: ***Errors in the energy in a strongly perturbed binary system for different mathematical schemes. The plot was taken from [17] as an example of the accuracy of other numerical integrators. The curve labelling is same as in Figure 8.12.***

The plots depict the behavior of different numerical schemes when 4000 orbital periods are computed. This is the same number of orbits as in the examples for the Runge-Kutta algorithm (see plots in 8.1). It shows that in all cases the error depend strongly on the applied algorithm. The plots labelled with H1, H2 and H3 were generated with so-called Hermite schemes, where a higher number indicates higher accuracy (the number of iteration

steps increases from H1 to H3). The plots with the index c1 and c2 were produced with a new method using Stumpff's c-functions. A detailed description of the underlying computations is given in *[17]*.

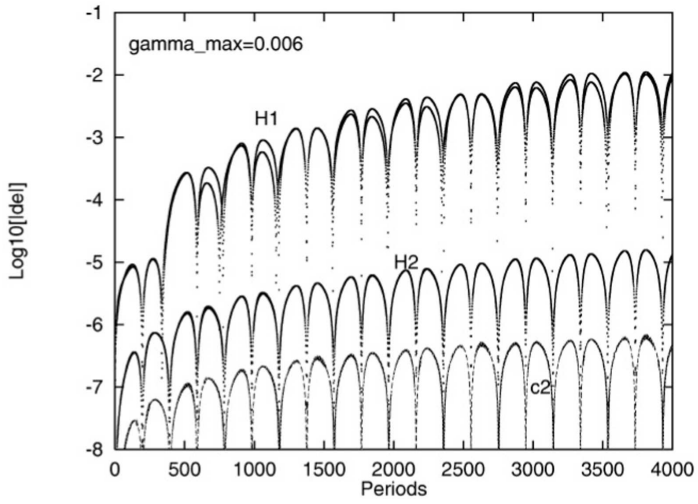

Figure 8.14: ***Errors in the eccentricity in a moderately perturbed binary system for methods with and without Stumpff functions. The plot was taken from [17] as an example of the accuracy of other numerical integrators. The curve labelling is same as in Figure 8.12.***

By comparing these results with those obtained with the 8^{th}-order Runge-Kutta algorithm it shows, that for the highly eccentric orbits the errors of the Runge-Kutta scheme have approximately the same order of magnitude ($10^{-2}-10^{-3}$) as the H1 scheme in Figures 8.13 and 8.14. Only in Figure 8.12 the results of the H1 approach are slightly better. For the less eccentric orbits the errors of the Runge-Kutta scheme are smaller ($10^{-5}-10^{-6}$) and they correspond more to those of the H2 or the c1 approach in Figures 8.13 and 8.14. Again, in Figure 8.12 the errors for the H2 and c1 scheme are slightly smaller. In most plots of the Runge-Kutta algorithm the errors increase constantly in time, which is also true for the plots here, if the mean values of the errors are considered (except for c2 in Figure 8.12, where the error does oscillate but not increase).

The comparison reveals, that the accuracy and stability of the 8^{th}-order Runge-Kutta algorithm is comparable to the performance of other numerical integrator schemes used for astrophysical calculations. An advantage of the Runge-Kutta algorithm might be, that only the current acceleration (e.g. the current forces acting on the spacecraft) need to be evaluated. For the Hermite approach, for instance, also the time derivative of the current acceleration is required, being sometimes difficult to compute. Hence, the accuracy, stability and also the uncomplicated implementation justify the use of the 8^{th}-order Runge-Kutta algorithm for the computation of spacecraft trajectories.

Having discussed and compared the accuracy of the numerical algorithm in this chapter, the presentation and validation of the unified approach to the computation of spacecraft trajectories is completed. The following chapter draws some conclusions and summarizes the advantages of the unified formulation. Furthermore some proposals for enhancements of the software tool are discussed and an overview of possible future applications is given.

Chapter 9

Conclusions

9.1 Major results

This diploma thesis demonstrated the feasibility of a unified approach to the computation of orbit, re-entry and launch trajectories. Numerous examples have been provided illustrating that a unified physical and algorithmic treatment of all flight phases is possible. The proposed approach can be applied to and possibly improve a wide range of mission analysis computations, such as studies concerning

- the orbiting phase and long term orbit propagation of satellites,
- the atmospheric arc and pure re-entry trajectories,
- mission analysis for re-entry vehicles combining orbiting and re-entry phase, or
- launch trajectories.

In all these cases the unified approach offers many potential advantages:

- The set of differential equations covers all types of orbits, even equatorial and highly eccentric orbits, whereas an unmodified classical formulation can only be applied to non-equatorial orbits.
- For the computation of trajectories that consist of an orbiting and a re-entry phase a change of reference frame is no longer required as all flight phases are described by the presented unified approach.
- The six variables of motion allow for a better understanding of the orbit, the position and the motion of the spacecraft. They refer directly to the actual position of the spacecraft as it is seen from an observer on the ground, whereas the six classical orbital elements are less intuitive and require a change of reference frame for their interpretation.
- Flight maneuvers needed for a controlled re-entry of a spacecraft can be planned long in advance. This is important in order to reach a specific entry interface and the correct flight envelope.
- The most important physical orbit perturbations can be implemented easily. The atmospheric lift and drag as well as the thrust are inherent in the equations.

9.2 Possible enhancements of the software tool

Although the results are plausible, the presented computer program could be enhanced by implementing the following features:

- Since the empirical atmosphere models will become more and more accurate and reliable, the accuracy of long-term predictions could be improved by implementing the latest versions of these models.
- For long-term orbit propagations additional physical forces (e.g. third body perturbations as mentioned in chapter 4) could be considered.
- Other perturbations (e.g. exospheric winds) could be taken into account.
- The spherical shape of the Earth could be replaced by an oblate geoid.

9.3 Future work

On the basis of the presented physical approach various future applications and developments can be considered. These might include the following:

- The code could be extended to a so-called 6-degree-of-freedom tool enabling three dimensional in-flight maneuvers for controlled, navigated and guided trajectories and re-entries.
- Presently most optimization procedures for trajectories are based on formulations assuming an inertial and therefore non-rotating reference frame (e.g. *[23]* and *[24]*). However, some optimization methods (e.g. Pontryagin's Maximum Principle) should also be applicable to the presented formulation. Consequently, the application of those methods could be investigated in order to optimize launch and re-entry trajectories under different constraints.
- The software tool could be used for planning interplanetary missions. Models for the gravitational fields and atmospheres of other planets could be included. In this case the program could be adapted to the simulation of missions to those planets, eventually including aerobraking and aerocapture maneuvers. These maneuvers use the atmospheric friction to bring a spacecraft from an interplanetary trajectory to a specified orbit around a planet.

The unified physical and algorithmic formulation and the implemented software program might just be the first step in a new direction of the development of mission analysis tools, as both combine a wide range of advantages with a solid basis for possible future activities and applications.

Acknowledgments

I would like to thank Professor Dr. Rainer Spurzem for giving me the unique opportunity to carry out this diploma thesis at the Astronomisches Rechen-Institut (ARI) in Heidelberg in cooperation with the European Space Agency (ESA). Since I am still dreaming of flying to space myself I am very grateful for having had this chance.

I would like to extend my thanks to Mr Marco Caporicci, Head of Human Transportation and Re-entry Systems Division at the European Space Research and Technology Center (ESTEC), for giving me the chance to work on the topic of trajectory computations within his division.

I wish to express my sincere gratitude to Mr Rafael Molina, Head of Aerodynamics and Aerothermodynamics Unit, who encouraged me in writing this diploma thesis during my stay at ESTEC. I frequently benefited from fruitful discussions with him concerning the content and the style of this work.

Furthermore, I would like to thank Dr. Andreas Just (ARI) for his precious comments and suggestions helping me write, describe and interpret more precisely.

Special thanks goes to Mr Mike Steinkopf, Head of Avionics and Operational Section (ESTEC), and Mr Olivier Bayle (ESTEC) for providing data used for the validation of the software tool.

Last but not least I want to thank Ms Jody Kirchner, Mr Daniel Heck and Mr Niels Rumpf who helped significantly improve the style of this diploma thesis.

I am most grateful to all the above!

Sascha P. Quanz

Appendix A

References

[1] ***Balmino, G. (1980):***
"Le Movement Elliptique Pertubé" in "Le Movement du Véhicule Spatial en Orbite"

[2] ***Escobal, P.R. (1965):***
"Methods of Orbit Determination"

[3] ***Nguyen X. Vinh, Adolf Busemann, Robert D. Culp (1980):***
"Hyersonic and Planetary Entry Flight Mechanics"

[4] ***M. Crouzeix, A.L. Mignot:***
"Analyse numérique des équations différentielles"

[5] *http://www.first.gmd.de/persons/tj/diss/node64.html*

[6] *http://nssdc.gsfc.nasa.gov/space/model/atmos/nrlmsise00.html*

[7] ***P. Fortescue, J. Stark (Editors):***
"Spacecraft systems engineering (second edition)"

[8] ***CNES, Cepaduès (1995):***
"Spaceflight Dynamics Part II"

[9] *http://www.heavens-above.com/ssorbitdecay.asp?*

[10] ***J.L. Lean, J.M. Picone, A. Hedin, S. Knowles, G. Moore:***
"Validating NRLMSIS Using Atmospheric Densities derived from Spacecraft Drag: Starshine Example"

[11] ***James R. Wertz, Wiley J. Larson (Editors):***
"Space Mission Analysis and Design (third edition)"

[12] ***D. Marty (1986):***
"Conception des véhicules spatiaux"

[13] *http://www.heavens-above.com/ss2-orbitdecayplot.asp*

[14] ***Oliver Montenbruck, Eberhard Gill:***
"Satellite Orbits - Models, Methods and Applications"
(Springer Verlag 2001)

[15] ***Starsem (The Soyuz Company):***
"Soyuz User's Manual (Issue No.3 April 2001)"

[16] ***Arianespace:***
"Ariane 4 User's Manual (Issue No.2 Feb. 1999)"

[17] ***Seppo Mikkola, Sverre J. Aarseth:***
"An efficient integration method for binaries in N-body simulations"
(New Astronomy 3 (1998) 309-320)

[18] *http://artemmis.univ-mrs.fr/cybermeca/Formcont/mecaspa/PROJETS/LANCEUR/*

[19] ***TSNIIMASH:***
"Integrated Software Packages for Soyuz TM, Progress M Reentry Computations"
(Work under ESA contract No. 10935/94/F/BM)

[20] ***EADS Launch Vehicles:***
"Launch of the Soyuz-TMA Spacecraft from French Guiana"
(Ref. No.: LR5 043486)

[21] ***E. Messerschmid, S. Fasoulas:***
"Raumfahrtsysteme"
(Springer Verlag 2001)

[22] *http://www.astronautix.com/lvs/index.htm*

[23] ***Anthony J. Calise, Nahum Melamed, Seungjae Lee:***
"Design and Evaluation of a Three-Dimensional Optimal Ascent Guidance Algorithm"
(Journal of Guidance, Control and Dynamics; Vol.21, No.6, Nov/Dec 1998)

[24] ***O. von Stryk, R. Burlisch***
"Direct and Indirect Methods for Trajectory Optimization"
(Annals of Operations Research 37(1992)357-373)

[25] ***Wilbur L. Hankey***
"Re-Entry Aerodynamics"
(AIAA Education Series 1988)

[26] *http://azinet.com/starshine/index.html*

[27] ***Aarseth, S.J.***
"Gravitational N-Body Simulations: Tools and Algorithms"
(Cambridge Monographs on Mathematical Physics, Cambridge University Press; England, Nov. 2003)

Appendix B

Nomenclature

$\vec{A}$	aerodynamic force
a	semi-major axis
$\vec{a}$	acceleration
α	angle of attack
b	semi-minor axis
c	distance from the center of an ellipse to one focus
c	specific gas constant
C_A	aerodynamic coefficient related to the axial force
C_D	drag coefficient
C_E	center of the Earth
C_g	center of gravity of the spacecraft
C_L	lift coefficient
C_N	aerodynamic coefficient related to the normal force
C_S	aerodynamic coefficient related to the side force
C_{nm}	sectoral harmonic coefficients of the Earth
χ	azimuth of the relative velocity
χ_0	initial azimuth of the relative velocity
$\vec{D}$	drag force
E	total energy
e	eccentricity
$\vec{e}$	Runge-Lenz vector
$\vec{e}_i$	unit vector pointing in i-direction
$\vec{F}_G$	gravitational force
$\vec{F}_N$	normal force (perpendicular to the velocity vector)
$\vec{F}_T$	tangential force (aligned with velocity vector)
$g(r)$	Earth acceleration
g_0	Earth acceleration on ground level
$g_{load,N}$	acceleration in normal direction in units of the Earth acceleration
$g_{load,T}$	acceleration in tangential direction in units of the Earth acceleration
γ	flight path angle
γ_0	initial flight path angle

γ_G	gravitational constant
γ_{deo}	flight path angle at entry interface
i	inclination of the orbit relative to the Earth
J_n	zonal harmonic coefficients of the Earth
κ	ratio of specific heat coefficients c_P/c_V
L	geographic latitude
L_0	initial geographic latitude
L_{deo}	latitude at entry interface
l	geographic longitude
l_0	initial geographic longitude
l_{deo}	longitude at entry interface
$\vec{L}$	lift force
M	mean anomaly
M	Mach number
m	mass
m_{ea}	mass of the Earth
$\vec{M}$	angular momentum
$\vec{M}_r$	reduced angular momentum
μ	bank angle
μ	viscosity of a fluid
μ_{ea}	gravitational parameter of the Earth
n	mean motion
Ω	right ascension of the ascending node
ω	argument of perigee
ω	angle between ΔV and V during de-orbit burn
$\vec{\Omega}$	angular velocity of the Earth
$\vec{\Omega}'$	angular velocity of the spacecraft centered coordinate system
p	pressure
P_{nm}	Legendre polynomials
ψ	heading
R_{ea}	Earth radius
r	altitude above the Earth surface/absolut value of radius vector
r_0	initial altitude above the Earth surface
r_{apo}	apoapsis/apogee
r_{deo}	altitude at entry interface
r_{peri}	periapsis/perigee
$\vec{r}$	radius vector/position vector
ρ	air density
ρ_0	air density at ground level
S	reference surface
S_{nm}	tesseral harmonic coefficients of the Earth
t	time
$\vec{T}$	thrust vector
T_s	specific impulse of a rocket engine
Θ	absolute temperature

ϑ_L	local pitch angle (related to the spacecraft)
ϑ_I	Galilean pitch angle (related to the launch pad)
U	gravitational potential
$\vec{V}$	velocity vector (relative to Earth)
V_0	initial relative velocity
V_s	speed of sound
V_{abs}^{apo}	absolute velocity at apogee
V_{abs}^{peri}	absolute velocity at perigee
X	horizontal distance from launch pad
$\dot{y}$	time derivative of the variable y
$\langle Z \rangle$	mean value of the variable Z

Appendix C

Acronyms and Reference Frames

LEO	Low Earth Orbit
ISS	International Space Station
FPA	Flight Path Angle
AoA	Angle of Attack
DOY	Day of Year
UT	Universal Time
GLAT	Geographic Latitude
GLONG	Geographic Longitude
STL	Local Apparent Solar Time
AF107	81-day Average of 10.7 cm Solar Flux
F107	Daily Value of 10.7 cm Solar Flux
AP	Daily Value of Magnetic Index

(X-Y-Z)	planet centered fixed coordinate system
(I-J-K)	planet centered rotating coordinate system
(i-j-k)	spacecraft centered coordinate system linked to the radius vector
(i'-j'-k')	spacecraft center coordinate system linked to the velocity vector
(x-y-z)	spacecraft centered coordinate system linked to the spacecraft body axes

Appendix D

The Source Code

In the following the source code and examples of the input files described in chapter 5 are given. The source code consists of a main program and numerous important functions. Both is shown in the following. Concerning the data bases containing the values of the solar flux and the magnetic index of the last decades for the NRLMSISE-00 atmosphere model, only an example of the required format will be given.

D.1 The main program

```
/*-------------------------------------------------------------*/
/*  THIS FILE CONTAINS THE MAIN PROGRAM FOR CALCULATING ORBIT */
/*                PROPAGATION AND TRAJECTORIES                 */
/*-------------------------------------------------------------*/

#include <stdio.h>
#include <math.h>
#include <stdlib.h>
#include <assert.h>
#include "nrlmsise-00.h"
#include "functions.h"

/*-------------------------------------------------------------*/
/* ------------------- BEGIN OF MAIN ----------------------- */
/*-------------------------------------------------------------*/

int main() {
    extern int ap[54][366];
    /*arrays for magnetic index*/

    extern const double flux[54][366];
    /*arrays for solar flux*/

    double a[9][9],b[9];
    /*array for Runge-Kutta coefficients*/
```

```
double kr[9],kgamma[9],kL[9],kl[9],kchi[9],kV[9];
/*coefficients for variables of motion in Runge Kutta*/

double r[2],L[2],chi[2],V[2],gamma[2],l[2];
/*fields for manipulation of the main variables*/

double XX[2];
/*horizontal distance from liftoff for Runge-Kutta*/

double kXX[9],dXX,ddXX;
/*changes in the horizontal distance*/

double rho,rho0,my;
/*air density,air density at ground level,bank angle*/

double omega,apogee,perigee,v_abs;
/*angular velocity Earth,Apogee,Perigee,absolute velocity*/

double AF107,F107;
/*10.7cm flux 81day average and daily value*/

float SOLARTIME;
/*apparent local solar time*/

double S,Cd,Cl,m;
/*surface,drag coefficient,lift coefficient,mass*/

int Cd_flag, Cl_flag;
/*flags for defining input file for aerodynamic coefficients*/

double radius,g,printflag;
/*radius of the Earth,gravitational constant,printflag*/

int printflag2;
/*printflag 2: see physical_model_input.txt*/

double UT,time,t,PI;
/*Universal time, elapsed time, integrating time step, pi*/

double dr,dV,dgamma,dL,dl,dchi,ddr,ddV,ddl;
double ddL,ddgamma,ddchi;
/*changes in the variables*/

int i,j,q,s,AP,orbitcounter;
/*counters,magnetic index*/
```

```
int DOY,YEAR,maxDOY;
/*actual day of year,actual year,days within actual year*/

int atmos_flag,gravi_flag;
/*atmospheric model flag and gravitational field flag*/

int latitude_flag;
/*latitude flag:  see orbit_input.txt*/

double deorbit_V,inclination;
/*delta V for deorbit burn and inclination of orbit*/

int deorbit_flag, burn_indicator;
/*indicators for the deorbit burn*/

double nrlmsise_correction_factor = 0.85;
/*correction factor for density provided by NRLMSISE00*/

int direction_flag;
/*direction flag: see orbit_input.txt*/

int answer;
/*specifies the input file*/

float dummy;
/*just a dummy variable*/

double gravi_constant;
/*gravitational constant times mass of the Earth*/

int stop_type_flag;
/*flag to identify type of the loop limit*/

float loop_limit;
/*contains the limit for the loop (e.g. maximum time)*/

double F_N, F_T, T, alpha;
/*Normal force,tangential force,thrust,angle of attack*/

double temperature, theta_L, theta;
/*absolute temperature and pitch angles*/

double lift_off;
/*lift off time in sec after ignition*/

double speed_of_sound, mach;
/*speed of sound and mach number*/
```

```
    double g0, g_load_N, g_load_T;
    /*Earth acceleration, tangential and normal g-loads*/

    double Cd_AoA_array[30], Cd_MACH_array[30], Cd_array[30][30];
    /*arrays containing the values for the angle of attack,  */
    /*for the mach number and those for Cd. Needed to read   */
    /*from Cd_input if required                              */

    int Cd_amount_of_MACH, Cd_amount_of_AoA;
    /*amount of mach numbers and angle of attacks            */
    /*specified in Cd_input                                  */

    double Cl_AoA_array[30], Cl_MACH_array[30], Cl_array[30][30];
    int Cl_amount_of_MACH, Cl_amount_of_AoA;
    /*see above!     */

    int pitch_flag, amount_of_pitches;
    /*flag indicates whether local pitch or Galilean pitch    */
    /*amount refers to amount of interpolation functions      */
    /*for the pitch law                                       */

    double pitch_array[30][4];
    /*contains start and stop time and initial and end value  */
    /* of pitch for every interpolation function for the      */
    /*pitch law                                               */

/*-------------------------------------------------------------*/
/*-------------------------------------------------------------*/

/*coefficients for 8th-order Runge Kutta*/
    a[1][1]=0.; a[2][2]=0.; a[3][3]=0.; a[4][4]=0.;
    a[5][5]=0.; a[6][6]=0.; a[7][7]=0.; a[8][8]=0.;
    a[2][1]=1./6.;  a[3][1]=4./75.; a[3][2]=16./75.;
    a[4][1]=5./6.;  a[4][2]=-8./3.; a[4][3]=5./2.;
    a[5][1]=-8./5.; a[5][2]=144./25.;
    a[5][3]=-4.;  a[5][4]=16./25.;  a[6][1]=361./320.;
    a[6][2]=-18./5.; a[6][3]=407./128.;  a[6][4]=-11./80.;
    a[6][5]=55./128.;a[7][1]=-11./640.;  a[7][2]=0.;
    a[7][3]=11./256.;a[7][4]=-11./160.; a[7][5]=11./256.;
    a[7][6]=0.; a[8][1]=93./640.; a[8][2]=-18./5.;
    a[8][3]=803./256.; a[8][4]=-11./160.; a[8][5]=99./256.;
    a[8][6]=0.;a[8][7]=1.;

    b[1]=7./1408.; b[2]=0.; b[3]=1125./2816.; b[4]=9./32.;
    b[5]=125./768.; b[6]=0; b[7]=5./66.; b[8]=5./66.;
```

```
/*-------------------------------------------------------------*/
/*               The File-handles                              */
/*-------------------------------------------------------------*/

    FILE* output_total;
    FILE* output_total2;    //*file-handles for writing output*/
    FILE* output_total3;

    FILE* input;            //*file-handle for reading input  */
    FILE* spacecraft_input;
    FILE* physical_model_input;
    FILE* stop_condition_input;

    FILE* Cd_input;
    FILE* Cl_input;
    FILE* pitch_input;

/*-------------------------------------------------------------*/
/*             chose and check the input file                  */
/*-------------------------------------------------------------*/
    do
    {
        printf("\nWhich input file shall be considered:\n");
        printf("\n1: orbit_input.txt: You specified an orbit");
        printf("\n2: manual_input.txt: You specified the initial conditions");
        printf("\n3: launch_input.txt: You specified a launch configuration");
        printf("\n0: abort program\n");
        printf("\nPlease type the number of the desired input file:  ");
        scanf("%i",&answer);
    }while (answer != 1 && answer !=2 && answer !=3 && answer != 0);

    if (answer == 0)
    {
        exit(0);
    };
    if (answer == 1)
    {
        input = fopen ("orbit_input.txt","r");
    };
    if (answer == 2)
    {
        input = fopen ("manual_input.txt","r");
    };
    if (answer == 3)
    {
        input = fopen ("launch_input.txt","r");
    };
```

```
/*check whether input file is valid; if not: error message!*/

    if (input == 0)
        {
        printf("\nThe desired input file does not exist!
                  Please create input file and try again!\n");
        exit(0);
        };

/*count the number of entries in the input file.*/

    i=0;
    if (answer != 3)
    {
    while (fscanf(input,"%f",&dummy)==1)
        {
        i++;
        };
    };

/*check whether input file contains right amount of parameters!*/

    if ((answer == 1 && i!=7) || (answer ==2 && i!=7))
        {
        printf("Please check the input file! \n");
        printf("It does not contain the right amount of parameters!\n");
        exit(0);
        };

/*open input files containing spacecraft parameters and  */
/*physical model                                         */

    spacecraft_input = fopen ("spacecraft_input.txt","r");
    physical_model_input = fopen ("physical_model_input.txt","r");
    stop_condition_input = fopen ("stop_condition_input.txt","r");

    if ( spacecraft_input == 0 )
    {
        printf("The file spacecraft_input.txt could not be found!
               This file is needed for the execution of the program!\n");
        exit(0);
    };
    if ( physical_model_input == 0 )
        {
        printf("The file physical_model_input.txt could not be found!
               This file is needed for the execution of the program!\n");
```

```
        exit(0);
    };
    if ( stop_condition_input == 0 )
        {
        printf("The file stop_condition_input.txt could not be found!
               This file is needed for the execution of the program!\n");
        exit(0);
    };

/*-------------------------------------------------------------*/

/*count the number of entries in the spacecraft input file.*/

    i=0;
    while (fscanf(spacecraft_input,"%f",&dummy)==1)
        {
        i++;
        };

/*check whether input file contains right amount of parameters!*/

    if (i!=8)
        {
        printf("Please check the input file spacecraft_input.txt! \n");
        printf("It does not contain the right amount of parameters!\n");
        exit(0);
        };

/*count the number of entries in the physical model input file.*/

    i=0;
    while (fscanf(physical_model_input,"%f",&dummy)==1)
        {
        i++;
        };

/*check whether input file contains right amount of parameters!*/

    if (i!=10)
        {
        printf("Please check the input file physical_model_input.txt! \n");
        printf("It does not contain the right amount of parameters!\n");
        exit(0);
        };
```

```
/*count the number of entries in the stop condition input file.*/

    i=0;
    while (fscanf(stop_condition_input,"%f",&dummy)==1)
        {
        i++;
        };

/*check whether input file contains right amount of parameters!*/

    if (i!=2)
        {
        printf("Please check the input file stop_condition_input.txt! \n");
        printf("It does not contain the right amount of parameters!\n");
        exit(0);
        };

/*-------------------------------------------------------------*/
/*                 rewind all the input files!                 */
/*-------------------------------------------------------------*/

    rewind (spacecraft_input);
    rewind (physical_model_input);
    rewind (stop_condition_input);
    if (answer != 3)
        rewind (input);

/*-------------------------------------------------------------*/
/*                initialization of variables                  */
/*-------------------------------------------------------------*/

    PI=acos(-1);
    rho0=1.225;          /*air desity at ground level [kg/m^3]*/

/*-------------------------------------------------------------*/
/*                  READ PHYSICAL MODEL PARAMETERS             */
/*------------------------------------------------------------ */

    fscanf(physical_model_input,"%lf",&radius);
    /*read Earth radius from input file*/
    fscanf(physical_model_input,"%lf",&omega);
    /*read Angular Velocity from input file*/
    fscanf(physical_model_input,"%lf",&gravi_constant);
    /*read gravitational constant times earth mass from input file*/
    fscanf(physical_model_input,"%lf",&UT);
    /*read UT   from input file*/
    fscanf(physical_model_input,"%i",&DOY);
```

```
    /*read DOY  from input file*/
    fscanf(physical_model_input,"%i",&YEAR);
    /*read YEAR from input file*/
    fscanf(physical_model_input,"%i",&atmos_flag);
    /*read atmospheric flag from input*/

    fscanf(physical_model_input,"%i",&gravi_flag);
    /*read gravitational field flag from input*/

    fscanf(physical_model_input,"%lf",&t);
    /*read integration timestep*/

    fscanf(physical_model_input,"%i",&printflag2);
    /*read printflag2*/

    fclose(physical_model_input);

    g0=gravi_constant/(radius*radius);
    /*Earth acceleration [m/s^2]*/

/*-------------------------------------------------------------*/
/*                      READ SPACECERAFT PARAMETERS            */
/*-------------------------------------------------------------*/

    fscanf(spacecraft_input,"%lf",&m);
    /*read mass from input file*/
    fscanf(spacecraft_input,"%lf",&S);
    /*read reference area from input*/
    fscanf(spacecraft_input,"%i",&Cl_flag);
    /*read Cl_flag from input file*/
    fscanf(spacecraft_input,"%lf",&Cl);
    /*read Cl from input file*/
    fscanf(spacecraft_input,"%i",&Cd_flag);
    /*read Cl_flag from input file*/
    fscanf(spacecraft_input,"%lf",&Cd);
    /*read Cd from input file*/
    fscanf(spacecraft_input,"%lf",&my);
    /*read bank angle from input file*/
    fscanf(spacecraft_input,"%lf",&alpha);
    /*read angle of attack from input file*/

    my=my/180.*PI;
    alpha=alpha/180.*PI;

    fclose(spacecraft_input);
```

```
/*-------------------------------------------------------------*/
/*  If aerodynamic coefficients depend on MACH and AoA read   */
/*           data from extra input file                        */
/*-------------------------------------------------------------*/

/*1.) for the drag coefficient: */

    if (Cd_flag == 1)
    {
        Cd_input = fopen("Cd_input.txt","r");
        if ( Cd_input == 0 )
        {
            printf("The file 'Cd_input.txt' could not be found!
            This file is needed for the execution of the program!\n");
            exit(0);
        };

        fscanf(Cd_input,"%i",&Cd_amount_of_MACH);
        fscanf(Cd_input,"%i",&Cd_amount_of_AoA);

/*read the different MACH numbers provided in input file*/

        for (i=0;i<Cd_amount_of_MACH;i++)
        {
            fscanf(Cd_input,"%lf",&Cd_MACH_array[i]);
        };

/*read the different angle of attack provided in input file*/

        for (i=0;i<Cd_amount_of_AoA;i++)
        {
            fscanf(Cd_input,"%lf",&Cd_AoA_array[i]);
        };

/*read the different Cd values provided in input file into    */
/* MACH-AoA matrix                                           */

        for (i=0;i<Cd_amount_of_MACH;i++)
        {
            for (j=0;j<Cd_amount_of_AoA;j++)
            {
                fscanf(Cd_input,"%lf",&Cd_array[i][j]);
            };
        };
        fclose(Cd_input);
    };
```

```
/*-------------------------------------------------------------*/

/*2.)  for the lift coefficient: */
    if (Cl_flag == 1)
    {
        Cl_input = fopen("Cl_input.txt","r");
        if ( Cl_input == 0 )
        {
            printf("The file 'Cl_input.txt' could not be found!
            This file is needed for the execution of the program!\n");
            exit(0);
        };

        fscanf(Cl_input,"%i",&Cl_amount_of_MACH);
        fscanf(Cl_input,"%i",&Cl_amount_of_AoA);

/*read the different MACH numbers provided in input file*/

        for (i=0;i<Cl_amount_of_MACH;i++)
        {
            fscanf(Cl_input,"%lf",&Cl_MACH_array[i]);
        };

/*read the different angle of attack provided in input file*/

        for (i=0;i<Cl_amount_of_AoA;i++)
        {
            fscanf(Cl_input,"%lf",&Cl_AoA_array[i]);
        };

/*read the different Cl values provided in input file into    */
/*MACH-AoA matrix                                            */

        for (i=0;i<Cl_amount_of_MACH;i++)
        {
            for (j=0;j<Cl_amount_of_AoA;j++)
            {
                fscanf(Cl_input,"%lf",&Cl_array[i][j]);
            };
        };
        fclose(Cl_input);
    };
```

```
/*-------------------------------------------------------------*/
/*                       READ STOP CONDITION                   */
/*------------------------------------------------------------ */

    fscanf(stop_condition_input,"%i",&stop_type_flag);
    /*read stop type flag from input file*/
    fscanf(stop_condition_input,"%f",&loop_limit);
    /*read limit of the main loop area from input*/

    fclose(stop_condition_input);

/*-------------------------------------------------------------*/
/*        READING AND COMPUTATION OF THE INITIAL CONDITIONS    */
/*-------------------------------------------------------------*/

/*-------------------------------------------------------------*/
/*a) Manual input*/
/*-------------------------------------------------------------*/

    if(answer == 2)
    {
        fscanf(input,"%lf",&r[0]);
        fscanf(input,"%lf",&V[0]);
        fscanf(input,"%lf",&gamma[0]);
        fscanf(input,"%lf",&chi[0]);
        fscanf(input,"%lf",&L[0]);
        fscanf(input,"%lf",&l[0]);
        fscanf(input,"%i",&burn_indicator);
        r[0] = r[0] * 1000 + radius;
        gamma[0]=gamma[0]/180.*PI;
        l[0]=l[0]/180.*PI;
        L[0]=L[0]/180.*PI;
        chi[0]=chi[0]/180.*PI;

        fclose(input);
    };
/*-------------------------------------------------------------*/
/*b) Orbit input*/
/*-------------------------------------------------------------*/

    if(answer == 1)
    {
        fscanf(input,"%lf",&apogee);
        fscanf(input,"%lf",&perigee);
        fscanf(input,"%lf",&l[0]);
        fscanf(input,"%lf",&inclination);
        fscanf(input,"%i",&latitude_flag);
```

```
        fscanf(input,"%i",&direction_flag);
        fscanf(input,"%i",&burn_indicator);

        apogee = apogee * 1000 + radius;
        perigee = perigee * 1000 + radius;
        l[0]=l[0]/180.*PI;
        gamma[0]=0.;

        fclose(input);

/*-------------------------------------------------------------*/

/*Calculate initial velocity and azimuth depending on input   */

        if((latitude_flag == 1) && (direction_flag == 1))
/*start perigee minimum latitude going east*/
        {
            L[0]=-inclination/180.*PI;
            chi[0]=90./180.*PI;
            r[0]=perigee;
            v_abs = sqrt(gravi_constant/r[0])*sqrt(2*apogee/(apogee+r[0]));
            V[0] = v_abs - (omega*r[0])*cos(L[0]);
        };
        if((latitude_flag == 1) && (direction_flag == 2))
/*start perigee minimum latitude going west*/
        {
            L[0]=-inclination/180.*PI;
            chi[0]=270./180.*PI;
            r[0]=perigee;
            v_abs = sqrt(gravi_constant/r[0])*sqrt(2*apogee/(apogee+r[0]));
            V[0] = v_abs + (omega*r[0])*cos(L[0]);
        };
        if((latitude_flag == 2) && (direction_flag == 1))
/*start perigee maximum latitude going east*/
        {
            L[0]=inclination/180.*PI;
            chi[0]=90./180.*PI;
            r[0]=perigee;
            v_abs = sqrt(gravi_constant/r[0])*sqrt(2*apogee/(apogee+r[0]));
            V[0] = v_abs - (omega*r[0])*cos(L[0]);
        };
        if((latitude_flag == 2) && (direction_flag == 2))
/*start perigee maximum latitude going west*/
        {
            L[0]=inclination/180.*PI;
            chi[0]=270./180.*PI;
            r[0]=perigee;
```

```
            v_abs = sqrt(gravi_constant/r[0])*sqrt(2*apogee/(apogee+r[0]));
            V[0] = v_abs + (omega*r[0])*cos(L[0]);
        };

/*-------------------------------------------------------------*/

        if((latitude_flag == 3) && (direction_flag == 1))
/*start apogee minimum latitude going east*/
        {
            L[0]=-inclination/180.*PI;
            chi[0]=90./180.*PI;
            r[0]=apogee;
            v_abs = sqrt(gravi_constant/r[0])*sqrt(2*perigee/(perigee+r[0]));
            V[0] = v_abs - (omega*apogee)*cos(L[0]);
        };
        if((latitude_flag == 3) && (direction_flag == 2))
/*start apogee minimum latitude going west*/
        {
            L[0]=-inclination/180.*PI;
            chi[0]=270./180.*PI;
            r[0]=apogee;
            v_abs = sqrt(gravi_constant/r[0])*sqrt(2*perigee/(perigee+r[0]));
            V[0] = v_abs + (omega*apogee)*cos(L[0]);
        };
        if((latitude_flag == 4) && (direction_flag == 1))
/*start apogee maximum latitude going east*/
        {
            L[0]=inclination/180.*PI;
            chi[0]=90./180.*PI;
            r[0]=apogee;
            v_abs = sqrt(gravi_constant/r[0])*sqrt(2*perigee/(perigee+r[0]));
            V[0] = v_abs - (omega*apogee)*cos(L[0]);
        };
        if((latitude_flag == 4) && (direction_flag == 2))
/*start apogee maximum latitude going west*/
        {
            L[0]=inclination/180.*PI;
            chi[0]=270./180.*PI;
            r[0]=apogee;
            v_abs = sqrt(gravi_constant/r[0])*sqrt(2*perigee/(perigee+r[0]));
            V[0] = v_abs + (omega*apogee)*cos(L[0]);
        };
    };
```

```
/*-------------------------------------------------------------*/
/*c) Launch Input: only read latitude, longitude, desired     */
/*    inclination and lift-off time the rest is read by launch*/
/*    function!                                                */
/*-------------------------------------------------------------*/

    if (answer == 3)
    {
        fscanf(input,"%lf",&L[0]);
        fscanf(input,"%lf",&l[0]);
        fscanf(input,"%lf",&chi[0]);
        fscanf(input,"%lf",&lift_off);
        XX[0]=0.;
/*set horizontal distance to zero*/
        L[0]=L[0]*PI/180.;
        l[0]=l[0]*PI/180.;
        V[0]=0.01;
        r[0]=radius;
        gamma[0]=PI/2.;
        chi[0]=chi[0]*PI/180.;

        fclose(input);
    };

/*-------------------------------------------------------------*/
/*In case of launch trajectory read also the pitch-law from   */
/* input file!                                                 */
/*-------------------------------------------------------------*/

    if (answer == 3)
    {
        pitch_input = fopen("pitch_input.txt","r");
        if ( pitch_input == 0 )
        {
            printf("The file 'pitch_input.txt' could not be found!
            This file is needed for the execution of the program!\n");
            exit(0);
        };

        fscanf(pitch_input,"%i",&pitch_flag);
/* initial pitch = 0 */
/* local pitch = 1   */
        fscanf(pitch_input,"%i",&amount_of_pitches);
/*how many equations are about to follow?  */

        for (i=0;i<amount_of_pitches;i++)
        {
```

```
            for (j=0;j<4;j++)
                fscanf(pitch_input,"%lf",&pitch_array[i][j]);
        };

        fclose(pitch_input);
    };

/*-------------------------------------------------------------*/
/*calculation of actual solartime*/

    SOLARTIME=UT/3600+l[0]/15;
/*------------------------------------------------------------ */

/*------------------------------------------------------------ */
/*            END OF CALCULATING INITIAL CONDITIONS           */
/*-------------------------------------------------------------*/

/*opening output files*/
    output_total = fopen ("Results.txt","w");
    output_total2 = fopen ("Results2.txt","w");
    output_total3 = fopen ("Results3.txt","w");

/*------------------------------------------------------------ */
/*             Initial parameters are found                   */
/*          the numerical integrator loop can start           */
/*------------------------------------------------------------ */

    printf("\nInitial conditions are found!
            \nStarting Trajectory computation!\n");
    j=1;
    time=orbitcounter=0;

/*write initial values in output file*/

    fprintf(output_total,"%5i%5i%10.2f",YEAR,DOY,UT);
    fprintf(output_total,"%15.2lf  ",time);
    fprintf(output_total,"%14.4f  ",r[0]/1000-6378.13649);
    fprintf(output_total,"%14.4f   %8.4f  ",V[0],gamma[0]/PI*180);
    fprintf(output_total,"%8.4f  ",L[0]/PI*180);
    fprintf(output_total,"%8.4f  ",l[0]/PI*180);
    fprintf(output_total,"%8.4f\n",chi[0]/PI*180);
```

```
/*------------------------------------------------------------ */
/*     THE NUMERICAL INTEGRATOR LOOP of orbit propagation     */
/*------------------------------------------------------------ */

    do
        {

/*the following is only needed if NRLMSISE00 atmosphere model */
/* is chosen!                                                 */

        if (atmos_flag == 2)
        {
            if (YEAR > 2013)

/*correction of the year for future missions with regard to   */
/*the 11-year solar circle                                    */

            {
                while (YEAR > 2013)
                    {
                    YEAR = YEAR - 11;
                    };
            };

            if (YEAR % 4 == 0)

/*calculation of the number of days of the actual year */

            {
                maxDOY = 366;
                if (YEAR % 100 == 0  && YEAR % 400 != 0)
                    maxDOY = 365;
            }
            else
                maxDOY = 365;

/*calculation of the 81-flux-average*/

            if (YEAR==2013 && maxDOY-DOY<41)
            {
                AF107=flux[YEAR-1960][DOY-1];
                for (s=1;s<=40;s++)
                    {
                    AF107=AF107+flux[YEAR-1960-11][DOY-1+s];

                    AF107=AF107+flux[YEAR-1960][DOY-1-s];
                    };
```

```
                AF107=AF107/81;
            }
            else
            {
                AF107=flux[YEAR-1960][DOY-1];
                for (s=1;s<=40;s++)
                    {
                    AF107=AF107+flux[YEAR-1960][DOY-1+s];

                    AF107=AF107+flux[YEAR-1960][DOY-1-s];
                    };
                AF107=AF107/81;
            };

/*get value for the magnetic index*/
            AP = ap[YEAR-1960][DOY-1];
/*get value for 10.7 solar flux*/
            F107 = flux[YEAR-1960][DOY-2];

/*below 80km magnetic index and 10.7 solar flux need correction*/
            if ((r[0]-radius)/1000<80)
            {
                F107=AF107=150.;
                AP=4.;
            };
        };
/*------------------------------------------------------------ */
/*         the beginning of the mathematical scheme           */
/*------------------------------------------------------------ */

        r[1]=r[0];
        V[1]=V[0];
        l[1]=l[0];
        L[1]=L[0];
        chi[1]=chi[0];
        gamma[1]=gamma[0];
        if (answer == 3.)
            XX[1]=XX[0];

        i=0;

        time=j*t;

        for (q=1;q<=8;q++)
            {
/*select atmospheric model:*/
```

```
/*no atmospheric*/
                if (atmos_flag == 0)
                {
                    rho = 0;
                    temperature = 0.;
                };
/*analytical model*/
                if (atmos_flag == 1)
                {
                    if (r[1]<=6533195)
                        rho=rho0*exp((-900*(r[1]-6378135))/6378135);
                    else
                        rho=3.5e-12*exp(-(((r[1]-6378135)/1000)-380)/48);
                    temperature = get_temperature (r[1],radius);
                };

/*NRLMSISE-00*/
                if (atmos_flag == 2)
                {
                    NRLMSISE(DOY,UT,(r[1]-radius)/1000,L[1],
                        l[1],SOLARTIME,AF107,F107,AP,&rho,&temperature);

                    rho=rho*1000.*nrlmsise_correction_factor;
                };

/*calculation of Earth acceleration:*/
/* symmetrical field */

                if (gravi_flag == 0)
                    g = gravi_constant/(r[1]*r[1]);

/*with J2 correction term*/

                if (gravi_flag == 1)
                    g = g_factor(L[1],l[1],r[1]);

/*-------------------------------------------------------------*/

/*compute mach number/*

                speed_of_sound=get_speed_of_sound(temperature);
                mach=V[1]/speed_of_sound;

/*get Cd and Cl from input files if necessary*/

                if (Cd_flag == 1)
                    Cd=get_Cd3(mach,alpha,Cd_AoA_array,Cd_MACH_array,
```

```
                    Cd_array,Cd_amount_of_MACH,Cd_amount_of_AoA);
            if (Cl_flag == 1)
                Cl=get_Cl3(mach,alpha,Cl_AoA_array,Cl_MACH_array,
                    Cl_array,Cl_amount_of_MACH,Cl_amount_of_AoA);

            if (answer == 3 && time < lift_off)
/*No changes whatsoever!  Therefore go on */

            {
                get_thrust_mass(time,&T,&m);
                F_N=0.;
                F_T=0;
                goto here;
            };

            if (answer == 3 && time >= lift_off)
            {

/*compute  pitch law!*/
                theta = get_theta_L2(time, amount_of_pitches,pitch_array);

                if (pitch_flag == 0)
/*galilean reference frame! transform to local reference frame*/

                    theta_L = theta*PI/180.+ XX[1]/radius + omega*(time-lift_off);
                else
                    theta_L = theta*PI/180.;

/*compute thrust and mass*/
                get_thrust_mass(time,&T,&m);
            };

            if (answer == 3)
            {
/*the forces*/
                F_N = T*sin(theta_L-gamma[1]) + 0.5*rho*S*Cl*V[1]*V[1];
                F_T = T*cos(theta_L-gamma[1]) - 0.5*rho*S*Cd*V[1]*V[1];
            }
            else
            {
                F_N = T*sin(alpha) + 0.5*rho*S*Cl*V[1]*V[1];
                F_T = T*cos(alpha) - 0.5*rho*S*Cd*V[1]*V[1];
            };

/*the g-loads*/
            g_load_N=F_N/m/g0;
            g_load_T=F_T/m/g0;
```

```
/*------------------------------------------------------------ */
/*------------------------------------------------------------ */

/*calculations of changes in six variables                     */
/*(the "real" runge kutta starts here!)                        */

                dr = delta_r(V[1],gamma[1]);

                dV = delta_V(V[1],gamma[1],chi[1],
                    r[1],L[1],g,m,omega,my,F_T);

                dl = delta_l(V[1],gamma[1],chi[1],r[1],L[1]);

                if (answer == 3 && theta == 90.)
                    dgamma=0;
                else
                    dgamma = delta_gamma(V[1],gamma[1],chi[1],
                        r[1],L[1],g,F_N,m,omega,my);

                dL = delta_L(V[1],gamma[1],chi[1],r[1]);

                if (answer == 3 && abs(gamma[1]-PI/2.)<1e-12)
                    dchi = 0.;

                else
                    dchi = delta_chi(V[1],gamma[1],chi[1],
                        r[1],L[1],g,m,omega,F_N,my);

                kr[q] = t * dr;
                kV[q] = t * dV;
                kl[q] = t * dl;
                kL[q] = t * dL;
                kchi[q] = t * dchi;
                kgamma[q] = t * dgamma;

                if (answer == 3)
                {
                    dXX = delta_X(radius,V[1],gamma[1],r[1]);
                    kXX[q] = t * dXX;
                };

                r[1]=r[0];
                V[1]=V[0];
                l[1]=l[0];
                L[1]=L[0];
                chi[1]=chi[0];
```

```
                gamma[1]=gamma[0];
                if (answer == 3)
                    XX[1]=XX[0];

                for (s=1;s<=q;s++)
                {
                    r[1]=r[1] + kr[s] * a[q+1][s];
                    V[1]=V[1] + kV[s] * a[q+1][s];
                    l[1]=l[1] + kl[s] * a[q+1][s];
                    L[1]=L[1] + kL[s] * a[q+1][s];
                    chi[1]=chi[1] + kchi[s] * a[q+1][s];
                    gamma[1]=gamma[1] + kgamma[s] * a[q+1][s];
                    if (answer ==3 )
                        XX[1]=XX[1] + kXX[s] * a[q+1][s];
                };
            };

        ddXX=ddr=ddV=ddl=ddL=ddgamma=ddchi=0;

        for (q=1;q<=8;q++)
            {
            ddV = ddV + kV[q] * b[q];
            ddr = ddr + kr[q] * b[q];
            ddl = ddl + kl[q] * b[q];
            ddL = ddL + kL[q] * b[q];
            ddchi = ddchi + kchi[q] * b[q];
            ddgamma = ddgamma + kgamma[q] * b[q];
            if (answer == 3)
                ddXX = ddXX + kXX[q] * b[q];
            };

/*new values after one integration step*/
        r[1] = r[0] + ddr;
        V[1] = V[0] + ddV;
        l[1] = l[0] + ddl;
        L[1] = L[0] + ddL;

        chi[1] = chi[0] + ddchi;
        gamma[1] = gamma[0] + ddgamma;
        if (answer == 3)
        {
            XX[1] = XX[0] + ddXX;
            XX[0]=XX[1];
        };

        r[0]=r[1];
        V[0]=V[1];
```

```
        l[0]=l[1];
        if (l[0]>=2*PI)
/*reset of longitude if >360*/
            {
            l[0]=l[0]-2*PI;
            orbitcounter=orbitcounter+1;
            };

        if (answer != 3)
/*during launch we allow counting backwards                    */
/*else we want to have positive longitude                      */
        {
            if (l[0]<0)
                {
                l[0]=2*PI+l[0];
                };
        };
        L[0]=L[1];
        chi[0]=chi[1];
        gamma[0]=gamma[1];

/*-------------------------------------------------------------- */
/* if de-orbit maneuver is to be calculated (burn_indicator=1)  */
/* calculate the delta V that is needed check where one would   */
/* end up if burn is initiated right now. If target can be      */
/* reached (deorbit_flag=1): initiate de-orbit.                 */
/* If target cannot be reached: no burn, wait until next        */
/* iteration step                                               */
/*--------------------------------------------------------------*/

        if (burn_indicator == 1)
            {
            deorbit_V = deorbit(radius,r[0],gamma[0]);

            deorbit_flag = landspotpredict(r[0],V[0],gamma[0],
                l[0],L[0],chi[0],deorbit_V,S,m,Cd,Cl,my,gravi_flag);
            };

        if (deorbit_flag == 1)
            {
            V[0]=V[0]-deorbit_V;
            burn_indicator=0;
            deorbit_flag=0;
            printf("\nDe-orbit-burn initiated! Delta V:\t%f\n\n",deorbit_V);
            };
/*-------------------------------------------------------------- */
/*-------------------------------------------------------------- */
```

```
here:

/*update of parameters*/
        UT=UT+t;
        if (UT > 86400.)
            {
            UT = UT - 86400.;
            DOY = DOY + 1;
            };
        if (DOY > maxDOY)
            {
            DOY = 1;
            YEAR = YEAR + 1;
            };
        SOLARTIME=UT/3600+l[i]/15.;

/*chose how many results are written in output*/
        printflag=j%printflag2;
        if (printflag==0)
            {
            fprintf(output_total,"%5i%5i%10.2lf",YEAR,DOY,UT);
            fprintf(output_total,"%15.2lf  ",time);
            fprintf(output_total,"%14.4lf  ",r[0]/1000-6378.13649);
            fprintf(output_total,"%14.4lf   %8.4lf  ",V[0],gamma[0]/PI*180);
            fprintf(output_total,"%8.4lf  ",L[0]/PI*180);
            fprintf(output_total,"%8.4lf  ",l[0]/PI*180);
            fprintf(output_total,"%8.4lf  \n",chi[0]/PI*180);

            fprintf(output_total2,"%5i%5i%10.2lf  ",YEAR,DOY,UT);
            fprintf(output_total2,"%15.2lf  ",time);
            fprintf(output_total2,"%   4.4e   ",rho);
            fprintf(output_total2,"%8.4lf  ",temperature);
            fprintf(output_total2,"%6.3lf   ",mach);
            fprintf(output_total2,"%6.4lf   ",Cd);
            fprintf(output_total2,"%6.4lf   \n",Cl);

            fprintf(output_total3,"%5i%5i%10.2lf  ",YEAR,DOY,UT);
            fprintf(output_total3,"%15.2lf  ",time);
            fprintf(output_total3,"%   4.4e   ",F_N);
            fprintf(output_total3,"%   4.4e  ",F_T);
            fprintf(output_total3,"%6.4lf   ",g_load_N);
            fprintf(output_total3,"%6.4lf   ",g_load_T);
            fprintf(output_total3,"%14.3lf   ",m);
            fprintf(output_total3,"%14.3lf   \n",T);
            };
        j++;
```

```
/*------------------------------------------------------------ */
/* end of loop depending on stop_condition_input*/

        if (r[0]/1000-6378.13649 <= 0 && answer != 3)
        {
            printf("\nSpacecraft touched down!\n");
            break;
        };

        if (stop_type_flag == 1 && r[0]/1000-6378.13649 <= loop_limit)
        {
            printf("\nDesired altitude is reached!\n");
            break;
        };

        if (stop_type_flag == 2 && r[0]/1000-6378.13649 >= loop_limit)
        {
            printf("\nDesired altitude is reached!\n");
            break;
        };

        if (stop_type_flag == 3 && orbitcounter == loop_limit)
        {
            printf("\nDesired amount of orbits was computed!\n");
            break;
        };

        if (stop_type_flag == 4 && time >= loop_limit)
        {
            printf("\nDesired orbit period was computed!\n");
            break;
        };
    }while(1<2);

/*-------------------------------------------------------------*/
/*                   END OF MAIN LOOP                          */
/*------------------------------------------------------------ */
/*    end of propagation loop, depending on chosen condition  */
/*------------------------------------------------------------ */

  printf("Trajectory complete.\n\nElapsed Time: %f sec\n\n",time);
  return 0;
}
/*-------------------------------------------------------------*/
/* ---------------- END OF MAIN PROGRAM ----------------------*/
/*-------------------------------------------------------------*/
```

D.2 The functions

```
/*------------------------------------------------------------ */
/* -------- FUNCTION FOR CALCULATING THE DELTA V------------- */
/* -----------          FOR THE DEORBIT BURN       ---------- */
/* SO THAT THE SPECIFIED TAEM INTERFACE CAN BE REACHED        */
/*------------------------------------------------------------ */


/*parameter: Earth radius, actual altitude, actual FPA    */

double deorbit(double radius, double r0, double gamma_s)
{
    double my,omega,omega_prime,X,a,b,c,result;

    double PI=acos(-1),v_abs,Omega;

    double land_longi, land_lati , Latitude;

    float re,gamma_e;

    FILE* taem;

/*OPEN FILE AND READ ENTRY INTERFACE PARAMETERS*/

    taem = fopen ("Deorbit.txt","r");

    if (taem != 0)
        {
        fscanf(taem,"%f",&land_longi);  /*INTERFACE LONGITUDE */
        fscanf(taem,"%f",&land_lati);   /*INTERFACE LATITUDE  */
        fscanf(taem,"%f",&re);          /*INTERFACE ALTITUDE  */
        fscanf(taem,"%f",&gamma_e);     /*INTERFACE FPA       */
        }
    else
        {
        printf("File TAEM.txt not accessible !!!\n");
        exit(1);
        assert(0);
        };

    Latitude=land_lati*PI/180.;

    re=re*1000.+radius;

    gamma_e=gamma_e/180.*PI;
```

```
/*absolute velocity of circular orbit */
    v_abs = sqrt(3.9894065e14/r0);

/*gravitational parameter of the Earth*/
    my=3.9894065e14;

    omega=PI;              /*angle between deorbit thrust and */
                           /*initial velocity                 */
    omega_prime=0.;        /*angle between initial velocity   */
                           /*and velocity after burn          */

    Omega=7.27221e-5;      /*Earth's angular velocity         */

/*---------------------------------------------------------- */
/*           Solving a quadratic equation for delta V        */
/* (taken from CNES spaceflight dynamics part 2)             */
/*---------------------------------------------------------- */


    X=(r0*r0)/(re*re*cos(gamma_e)*cos(gamma_e));

    a=(X*(cos(omega_prime)*cos(omega_prime)
        *sin(omega+gamma_s)*sin(omega+gamma_s)-1)+1);

    b=2.*v_abs*cos(omega_prime)*(cos(omega)
        -X*cos(gamma_s)*cos(omega+gamma_s));

    c=2.*my*(1./re-1./r0)+v_abs*v_abs
        *(1-X*cos(gamma_s)*cos(gamma_s));

    result=(-b+sqrt(b*b-4.*a*c))/(2.*a);

    fclose(taem);
    return result;

    }
```

```
/*-------------------------------------------------------------- */
/*--------------------   NRLMSISE   ------------------------- */
/*--------FUNCTION FOR NRLMSISE-00 ATMOSPHERE MODEL-----------*/
/*  RETURNS THE DENSITY AND TEMPERATURE FOR A GIVEN TIME,     */
/*  A GIVEN ALTITUDE AND A GIVEN POSITION ABOVE THE SURFACE   */
/*                      OF THE EARTH                          */
/*-------------------------------------------------------------- */

void NRLMSISE(int doy,float sec,double alt,double g_lat,
               doubleg_long, double lst, double f107A,
               double f107, int ap, double* rho, double* temperature)
{
    struct nrlmsise_output output[17];
    struct nrlmsise_input input[17];
    struct nrlmsise_flags flags;
    struct ap_array aph;
    int i;

/* input values */
    for (i=0;i<7;i++)
        aph.a[i]=100;
    flags.switches[0]=0;
    for (i=1;i<24;i++)
        flags.switches[i]=1;
    for (i=0;i<1;i++) {
        input[i].doy=doy;        /*specified: Day Of Year     */
        input[i].year=0;         /*           Year            */
        input[i].sec=sec;        /*           UT of the day   */
        input[i].alt=alt;        /*           Altitude        */
        input[i].g_lat=g_lat;    /*           Latitude        */
        input[i].g_long=g_long;  /*           Longitude       */
        input[i].lst=lst;        /*           Local Solar Time*/
        input[i].f107A=f107A;    /*           10.7cm Average  */
        input[i].f107=f107;      /*           actual 10.7 cm  */
        input[i].ap=ap;          /*           actual AP index */
    }

    for (i=0;i<1;i++)
        gtd7d(&input[i], &flags, &output[i]);

/* returns the atmospheric density */
    *rho = output[0].d[5];

/*returns temperature */
    *temperature = output[0].t[1];

}
```

```
/*------------------------------------------------------------ */
/*         DIFFERENTIAL EQUATIONS DESCRIBING THE MOTION       */
/*                    OF THE SPACECRAFT                       */
/*------------------------------------------------------------ */

/*------------------------------------------------------------ */
/* ------------ FUNCTION for CHANGES IN ALTITUDE ------------ */
/*------------------------------------------------------------*/

double delta_r (double V_0, double gamma_0)
    {
    double dr;

    dr = V_0*sin(gamma_0);
    return dr;
    }

/*------------------------------------------------------------ */
/* ------------ FUNCTION for CHANGES IN LATITUDE ------------ */
/*------------------------------------------------------------ */

double delta_L (double V_0, double gamma_0, double chi_0,
                    double r_0)
    {
    double dL;

    dL = V_0/r_0*cos(gamma_0)*cos(chi_0) ;
    return dL;
    }

/*------------------------------------------------------------ */
/* -------------- FUNCTION for CHANGES IN LONGITUDE---------- */
/*------------------------------------------------------------ */

double delta_l (double V_0, double gamma_0, double chi_0,
                double r_0,  double L_0)
    {
    double dl;

    dl =  V_0/r_0*cos(gamma_0)*sin(chi_0)/cos(L_0);
    return dl;
    }
```

```
/*------------------------------------------------------------ */
/* ------------ FUNCTION for CHANGES IN VELOCITY------------ */
/*------------------------------------------------------------ */

double delta_V (double V_0, double gamma_0, double chi_0,
                double r_0, double L_0, double g, double m,
                double omega, double my, double F_T)
    {
    double dV;

    dV= -g*sin(gamma_0)+F_T/m+omega*omega*r_0*cos(L_0)
        *(sin(gamma_0)*cos(L_0)-cos(gamma_0)*sin(L_0)*cos(chi_0));
    return dV;
    }

/*------------------------------------------------------------ */
/* ----------------- FUNCTION for CHANGES IN FPA ------------ */
/*------------------------------------------------------------ */

double delta_gamma (double V_0, double gamma_0, double chi_0,
                    double r_0, double L_0, double g, double F_N,
                    double m, double omega, double my )
    {
    double dgamma;
    dgamma =   -g/V_0*cos(gamma_0)+F_N/m*cos(my)/V_0+
        V_0/r_0*cos(gamma_0)+2.*omega*cos(L_0)*sin(chi_0)+
        1./V_0*omega*omega*r_0*cos(L_0)*(cos(gamma_0)*cos(L_0)
            +sin(gamma_0)*sin(L_0)*cos(chi_0)) ;

    return dgamma;
    }

/*------------------------------------------------------------*/
/*------------ FUNCTION for CHANGES IN AZIMUTH-------------- */
/*------------------------------------------------------------ */

double delta_chi (double V_0, double gamma_0, double chi_0,
                    double r_0, double L_0, double g, double m,
                    double omega, double F_N, double my)
    {
    double dchi;
    dchi = -F_N/m*sin(my)/V_0/cos(gamma_0)
            +V_0/r_0*cos(gamma_0)*tan(L_0)*sin(chi_0)
            + 2.*omega*(sin(L_0)-tan(gamma_0)*cos(L_0)*cos(chi_0))
            +omega*omega*r_0/V_0*sin(L_0)*cos(L_0)*sin(chi_0)/cos(gamma_0);
    return dchi;
    }
```

```
/*-------------------------------------------------------------*/
/*           correction of the gravitational potential         */
/*            due to oblateness of the Earth via J2 term       */
/* ----------------------------------------------------------- */

double g_factor (double latitude, double longitude, double r_i)
{
    double radius =6378136.49, my=3.9894065e14;

/*correction factors*/

    double J2 = 1082.63e-6;
    double C22 = 1.57e-6, S22 = -0.904e-6;
    double P20 = 0.5*(3*cos(latitude)*cos(latitude)-1);
    double P22 = 3*(1 - cos(latitude)*cos(latitude));

/*resulting Earth acceleration*/

    double g;

    g = my / (r_i*r_i) - 3*my*radius*radius / (r_i*r_i*r_i*r_i) *
        ( J2*P20 + P22*(C22*cos(2*longitude)+S22*sin(2*longitude)));

    return g;
}

/* ----------------------------------------------------------- */
/* --------- FUNCTION for computation of temperature -------- */
/* ----------------- using an analytical model ------------- */
/* ----------------------------------------------------------- */

double get_temperature (double altitude, double radius)
{
    double temperature;

/*convert to km above surface*/
    altitude=(altitude-radius)/1000.;

    if (altitude < 11.)
        temperature = -6.3636*altitude + 287.;

    if (altitude >= 11. && altitude < 15.)
        temperature = -0.25 * altitude + 219.75;

    if (altitude >= 15. && altitude < 25.)
        temperature =  0.1*altitude + 214.5;
```

```
    if (altitude >= 25. && altitude < 48.)
        temperature = 2.8261*altitude + 146.35;

    if (altitude >= 48. && altitude < 50.)
        temperature = 0.5*altitude + 258.;

    if (altitude >= 50. && altitude < 53.)
        temperature = -1./3.*altitude + 299.67;

    if (altitude >= 53. && altitude < 80.)
        temperature = -4.2963*altitude + 509.7;

    if (altitude >= 80. && altitude < 85.)
        temperature = -1.2*altitude + 262.;

    if (altitude >= 85. && altitude < 97.)
        temperature =  0.5*altitude + 117.5;

    if (altitude >= 97. && altitude < 170.)
        temperature = 8.6849*altitude - 676.44;

    if (altitude >= 170. && altitude < 220.)
        temperature = 2.8*altitude + 324.;

    if (altitude >= 220. && altitude <= 500.)
        temperature = 0.2143*altitude + 892.86;

    if (altitude > 500.)
    {
        printf("\n Sorry!\n For the actual altitude,
                there is no value for the temperature available!
                \n Please use NRLMSISE-00 atmophere model!\n");
        temperature = 0.;
        exit(0);
    }
    return temperature;
}
```

```
/*-------------------------------------------------------------*/
/*---------- FUNCTION for computation of speed of sound------ */
/*------------------------------------------------------------ */

double get_speed_of_sound (double temperature)
{
    double speed_of_sound;
/*Cp over Cv for air (specific heat coefficients*/
    float kappa=1.4;

/*specific gas constant unit:[J/(mol*K)]*/
    float gas_constant=8.31451;

/*constituents:N(78%),O2(21%),Ar(1%),unit[g]*/
    float mol_mass=28.96;

    speed_of_sound=sqrt(kappa*gas_constant*temperature/(mol_mass/1000.));
    return speed_of_sound;
}

/*-------------------------------------------------------------*/
/*-------------------HELP FUNCTION slope -------------------- */
/*computes the slope for a linear approxim. between 2 points  */
/*------------------------------------------------------------ */

double get_slope (double x1, double x2, double y1, double y2)
{
    double slope;
    if (y1 >= y2)
        slope=(y1-y2)/(x1-x2);
    if (y1 < y2)
        slope=(y2-y1)/(x2-x1);
    return slope;
}

/*------------------------------------------------------------ */
/* ----- FUNCTION for changes in horizontal distance   ------ */
/* -- needed for launch, returns actual horizontal distance --*/
/* ------------------------------------------------------------ */

double delta_X (double radius, double V_0, double gamma_0,
                double r_0)
{
    double dXX;
    dXX = radius*V_0*cos(gamma_0)/r_0;
    return dXX;
}
```

```
/*------------------------------------------------------------ */
/* --- FUNCTION for computing the actual Thrust and Mass ---- */
/* ------------------- needed for launch! --------------------*/
/*------------------------------------------------------------ */

get_thrust_mass (double time, double* thrust, double* mass)

{
    int amount_of_stages;
    int additional_mass_flag;
    int i,j;
    double dummy, payload_mass;

/*so far 11 parameters for each stage have been identified    */
    int items_per_stage=11;

/*so far 5 parameters for having additional mass items have   */
/*been identified                                             */

    int items_per_additional_mass=5;

/*maximum of stages possible*/
    int max_stages=10;

/*maximum of additional mass items possible*/
    int max_additional_mass=10;

    double launch_array1[max_stages][items_per_stage];
    double launch_array2[max_additional_mass][items_per_additional_mass];
    double  PI=acos(-1);

    FILE* launch_input;

    launch_input = fopen("launch_input.txt","r");
    if ( launch_input == 0 )
    {
        printf("The file 'launch_input.txt' could not be found!
               This file is needed for the execution of the program!\n");
        exit(0);
    };

/*Read variables that are not needed into dummy*/
    for (i=0; i<4; i++)
    {
        fscanf(launch_input,"%f",&dummy);
    };
```

```
    fscanf(launch_input,"%i",&amount_of_stages);
    fscanf(launch_input,"%lf",&payload_mass);
    fscanf(launch_input,"%i",&additional_mass_flag);


/*read the input from "launch_input.txt"                        */
/* a) Read input for each stage:                                */

    for (i=0; i<amount_of_stages; i++)
    {
/*number of engines*/
        fscanf(launch_input,"%lf",&launch_array1[i][0]);
/*thrust per engine*/
        fscanf(launch_input,"%lf",&launch_array1[i][1]);
/*thrust angle*/
        fscanf(launch_input,"%lf",&launch_array1[i][2]);
/*specific impulse*/
        fscanf(launch_input,"%lf",&launch_array1[i][3]);
/*burn start time*/
        fscanf(launch_input,"%lf",&launch_array1[i][4]);
/*burn stop time*/
        fscanf(launch_input,"%lf",&launch_array1[i][5]);
/*propellant mass*/
        fscanf(launch_input,"%lf",&launch_array1[i][6]);
/*mass burnt per time [kg/sec]*/
        launch_array1[i][7]=launch_array1[i][0]*launch_array1[i][1]
                        /launch_array1[i][3];
/*actual propellant mass*/
        launch_array1[i][8]=launch_array1[i][6];
/*separation time*/
        fscanf(launch_input,"%lf",&launch_array1[i][9]);
/*separation mass*/
        fscanf(launch_input,"%lf",&launch_array1[i][10]);
    };


/* a) Read input for additional mass:       */
    if (additional_mass_flag != 0)
    {
        for (i=0; i<additional_mass_flag; i++)
        {
/*mass*/
            fscanf(launch_input,"%lf",&launch_array2[i][0]);
/*start time*/
            fscanf(launch_input,"%lf",&launch_array2[i][1]);
/*stop time*/
            fscanf(launch_input,"%lf",&launch_array2[i][2]);
```

```
            if ((launch_array2[i][2]-launch_array2[i][1]) != 0.)
/*mass lost per time [kg/sec]*/
            launch_array2[i][3]=launch_array2[i][0]/
                                (launch_array2[i][2]-launch_array2[i][1]);

            else
            launch_array2[i][3]=0.;
/*actual mass*/
            launch_array2[i][4]=launch_array2[i][0];
        };
    };

    fclose (launch_input);

/*Calculate the actual thrust and mass*/

    *mass=*thrust=0.;

/*first we consider the boosters:*/

    for (i=0; i<amount_of_stages; i++)
    {
        if (time < launch_array1[i][4])
            *mass = *mass + launch_array1[i][6] + launch_array1[i][10];

/*time is during the firing interval*/
        if (time >= launch_array1[i][4] && time < launch_array1[i][5])
        {
            *thrust = *thrust + launch_array1[i][0]*launch_array1[i][1]
                            *cos(launch_array1[i][2]*PI/180.);

            launch_array1[i][8]=launch_array1[i][6]-(time-launch_array1[i][4])
                                *launch_array1[i][7];

            *mass= *mass + launch_array1[i][8] + launch_array1[i][10];
        };

/*time is after firing but before separation*/
        if (time >= launch_array1[i][5] && time < launch_array1[i][9])
        {
            launch_array1[i][8]=launch_array1[i][6]-(launch_array1[i][5]-
                launch_array1[i][4])*launch_array1[i][7];

            *mass= *mass + launch_array1[i][8] + launch_array1[i][10];
        };
    };
```

```
/*Now we take into account the additional mass*/

    if (additional_mass_flag != 0)
    {
        for (i=0; i<additional_mass_flag; i++)
        {
            if (launch_array2[i][3] != 0.)
            {
                if (time < launch_array2[i][1])
                    *mass = *mass + launch_array2[i][0];

                if(time >= launch_array2[i][1] && time < launch_array2[i][2])
                {
                    launch_array2[i][4]=launch_array2[i][0]-
                                (time-launch_array2[i][1])*launch_array2[i][3];

                    *mass = *mass + launch_array2[i][4];
                };
            };

            if (launch_array2[i][3] == 0. && time < launch_array2[i][1])
                *mass = *mass + launch_array2[i][0];
        };
    };

/*Finally we should add the payload mass*/

    *mass = *mass + payload_mass;
}

/*------------------------------------------------------------ */
/* -------- FUNCTION for the computation of Cd as  ---------- */
/*                    function of AoA and MACH                */
/*------------------------------------------------------------ */

double get_Cd3(double mach, double AoA, double AoA_array[30],
               double MACH_array[30], double Cd_array[30][30],
               int amount_of_MACH, int amount_of_AoA)

/*This function receives  arrays that contain the necessary   */
/*information about Cd as a function of MACH number and angle */
/*of attack                                                    */

{
    int AoA_left, AoA_right, mach_left, mach_right;
    double u,v;
```

```
/*help variables for the interpolation of the Cd values*/
    double Cd;
    int i,j;

    AoA=AoA*180/acos(-1);

/*if there is no Cd for the actual mach number or angle of    */
/*attack the program should be stopped                        */

    if (mach < MACH_array[0] || mach > MACH_array[amount_of_MACH-1] ||
        AoA < AoA_array[0] || AoA > AoA_array[amount_of_AoA-1])
    {
        printf("\nFor the actual mach number there is no
                                        Cd-coefficient available!");
        printf("\nThe program is stopped!");
        exit(0);
    }

/*check the MACH-AoA matrix for the position of the actual    */
/*values of mach number and angle of attack                   */

    for (i=1; i<=amount_of_MACH; i++)
    {
        if (mach <= MACH_array[i])
        {
            mach_left = i-1;
            mach_right = i;
            break;
        };
    };
    for (i=1; i<=amount_of_AoA; i++)
    {
        if (AoA <= AoA_array[i])
        {
            AoA_left = i-1;
            AoA_right = i;
            break;
        };
    };

/* 2-D Interpolation to get Cd */
    u= (Cd_array[mach_left][AoA_right]-Cd_array[mach_left][AoA_left])*
        (AoA-AoA_array[AoA_left])/(AoA_array[AoA_right]-AoA_array[AoA_left])+
        Cd_array[mach_left][AoA_left];

    v= (Cd_array[mach_right][AoA_right]-Cd_array[mach_right][AoA_left])*
        (AoA-AoA_array[AoA_left])/(AoA_array[AoA_right]-AoA_array[AoA_left])+
```

```
        Cd_array[mach_right][AoA_left];

    Cd= (v-u)*(mach-MACH_array[mach_left])/
        (MACH_array[mach_right]-MACH_array[mach_left])+u;

    return Cd;
}

/*-------------------------------------------------------------*/
/*----------- FUNCTION for the computation of Cl as ----------*/
/*                  function of AoA and MACH                   */
/*------------------------------------------------------------ */

double get_Cl3(double mach, double AoA, double AoA_array[30],
               double MACH_array[30], double Cl_array[30][30],
               int amount_of_MACH, int amount_of_AoA)

/*This function  receives arrays that contain the necessary   */
/*information about Cd as a function of MACH number and angle */
/*of attack                                                    */

{
    int AoA_left, AoA_right, mach_left, mach_right;
/*help variables for the interpolation of the Cd values*/
    double u,v;
    double Cl;
    int i,j;

    AoA=AoA*180/acos(-1);

/*if there is no Cl for the actual mach number or angle of    */
/*of attack the program should be stopped                      */

    if (mach < MACH_array[0] || mach > MACH_array[amount_of_MACH-1] ||
        AoA < AoA_array[0] || AoA > AoA_array[amount_of_AoA-1])
    {
        printf("\nFor the actual mach number there is no Cl-coefficient available!");
        printf("\nThe program is stopped!");
        exit(0);
    }

/*check the MACH-AoA matrix for the position of the actual    */
/*values of mach number and angle of attack                    */

    for (i=1; i<=amount_of_MACH; i++)
    {
        if (mach <= MACH_array[i])
```

```
        {
            mach_left = i-1;
            mach_right = i;
            break;
        };
    };
    for (i=1; i<=amount_of_AoA; i++)
    {
        if (AoA <= AoA_array[i])
        {
            AoA_left = i-1;
            AoA_right = i;
            break;
        };
    };

/* 2-D Interpolation to get Cd */
    u= (Cl_array[mach_left][AoA_right]-Cl_array[mach_left][AoA_left])*
        (AoA-AoA_array[AoA_left])/(AoA_array[AoA_right]-AoA_array[AoA_left])+
        Cl_array[mach_left][AoA_left];

    v= (Cl_array[mach_right][AoA_right]-Cl_array[mach_right][AoA_left])*
        (AoA-AoA_array[AoA_left])/(AoA_array[AoA_right]-AoA_array[AoA_left])+
        Cl_array[mach_right][AoA_left];

    Cl= (v-u)*(mach-MACH_array[mach_left])/
        (MACH_array[mach_right]-MACH_array[mach_left])+u;

    return Cl;
}

/*-------------------------------------------------------------*/
/* ---------------------- FUNCTION PITCH  ------------------- */
/* needed for launch,returns actual PITCH from pitch_input.txt*/
/*------------------------------------------------------------ */

double get_theta_L2 (double time, int amount_of_pitches,
                             double pitch_array[30][4])
{
    int i;
    double time_left, time_right, pitch_left, pitch_right;
    double slope, b;
    double theta;

    for (i=0; i<amount_of_pitches; i++)
    {
        if (pitch_array[i][0]<=time && pitch_array[i][1]>time)
```

```
        {
            time_left=pitch_array[i][0];
            time_right=pitch_array[i][1];
            pitch_left=pitch_array[i][2];
            pitch_right=pitch_array[i][3];

/*Interpolate between specified points and return pitch angle */

            slope=get_slope(time_left,time_right,pitch_left,pitch_right);
            b=pitch_left-(time_left*slope);
            theta = slope* time+b;
            return theta;
            break;
        };
    };
    printf("\n Could not find pitch! Please check pitch_input.txt!\n");
    return 0;
}

/*------------------------------------------------------------ */
/*       THIS FILE CONTAINS THE FUNCTION THAT CALCULATES      */
/*     THE DISCREPANCY BETWEEN THE SPECIFIED ENTRY INTERFACE  */
/*       AND THE POSITION THAT WOULD BE REACHED IF THE        */
/*           DE-ORBIT BURN TOOK PLACE IMMEDIATELY.            */
/*            IF THE DIFFERENCE IS SMALL ENOUGH               */
/*   (+/-50KM AT 120KM ALTITUDE)  THE COMMAND FOR THE BURN IS */
/*   GIVEN. FOR THE PREDICTION THE SAME ALGORITHM AS FOR THE  */
/*               MAIN PROGRAM IS APPLIED.                     */
/*------------------------------------------------------------ */

#include <stdio.h>
#include <math.h>
#include <stdlib.h>
#include <assert.h>
#include "functions2.h"

/*------------------------------------------------------------ */
/*------------------------------------------------------------ */

int landspotpredict(double r0, double V0, double gamma0,
                        double l0, double L0,double chi0,
                        double deorbit_V, double S, double m,
                        double Cd,double Cl, double bank, int gravi_flag)
{

/*8th order Runge-Kutta coefficients*/
    double a[9][9],b[9];
```

```
/*coefficients for each variable in Runge Kutta*/
    double kr[9],kgamma[9],kL[9],kl[9],kchi[9],kV[9];

/*fields for munipulation of each variable*/
    double r[2],L[2],chi[2],V[2],gamma[2],l[2];

/*air density,air density at ground level,bank angle*/
    double rho,rho0,my;

/*angular velocity Earth*/
    double omega;

/*radius of the Earth,gravitational constant*/
    double radius,g;

/*time passed by, intergrating time step, pi*/
    double time,t,PI;

/*changes in the variables*/
    double dr,dV,dgamma,dL,dl,dchi,ddr,ddV,ddl,ddL,ddgamma,ddchi;

/*counters*/
    int i,j,q,s;

    float x,taem_latitude, taem_longitude, taem_altitude, taem_fpa;

/*Normal force,tangential force,thrust,angle of attack*/
    double F_N, F_T, T, alpha;

    FILE* taem;
/*------------------------------------------------------------ */
    taem = fopen ("Deorbit.txt","r");

    if (taem != 0)
        {
        fscanf(taem,"%f",&taem_longitude);
        fscanf(taem,"%f",&taem_latitude);
        fscanf(taem,"%f",&taem_altitude);
        fscanf(taem,"%f",&taem_fpa);
        }
    else
        {
        printf("\nDeorbit.txt cannot be found!\n");
        exit(1);
        assert(0);
        };
```

```
/*coefficients for 8th-order Runge Kutta*/
    a[1][1]=0.; a[2][2]=0.; a[3][3]=0.; a[4][4]=0.;
    a[5][5]=0.; a[6][6]=0.; a[7][7]=0.; a[8][8]=0.;
    a[2][1]=1./6.;  a[3][1]=4./75.; a[3][2]=16./75.;
    a[4][1]=5./6.;  a[4][2]=-8./3.; a[4][3]=5./2.;
    a[5][1]=-8./5.; a[5][2]=144./25.; a[5][3]=-4.;
    a[5][4]=16./25.;a[6][1]=361./320.;a[6][2]=-18./5.;
    a[6][3]=407./128.; a[6][4]=-11./80.;a[6][5]=55./128.;
    a[7][1]=-11./640.; a[7][2]=0.; a[7][3]=11./256.;
    a[7][4]=-11./160.; a[7][5]=11./256.; a[7][6]=0.;
    a[8][1]=93./640.; a[8][2]=-18./5.; a[8][3]=803./256.;
    a[8][4]=-11./160.; a[8][5]=99./256.;a[8][6]=0.;a[8][7]=1.;

    b[1]=7./1408.; b[2]=0.; b[3]=1125./2816.; b[4]=9./32.;
    b[5]=125./768.; b[6]=0; b[7]=5./66.; b[8]=5./66.;

/*------------------------------------------------------------ */
/*---------------- Initialization of variables ---------------*/
/*------------------------------------------------------------ */

    radius=6378136.49;
    PI=acos(-1);
    rho0=1.225;
    omega=7.27221e-5;

    t=1.;

    j=1;
    time=0;

    /*Landing spot margin for Longitude*/

    x = 2.*PI/360.*radius/1000.*cos(taem_latitude/180.*PI);

    r[0]=r0;
    V[0]=V0-deorbit_V;
    l[0]=l0;
    L[0]=L0;
    gamma[0]=gamma0;
    chi[0]=chi0;

    my=bank;
```

```
/*------------------------------------------------------------ */
/*------- The numerical loop (Runge-Kutta algorithm) -------- */
/*------------------------------------------------------------ */

    do
        {
        r[1]=r[0];
        V[1]=V[0];
        l[1]=l[0];
        L[1]=L[0];
        chi[1]=chi[0];
        gamma[1]=gamma[0];

        i=0;

        time=j*t;

        for (q=1;q<=8;q++)
            {
            if (r[1]<=6533195)
                {
                rho=rho0*exp((-900*(r[1]-6378135))/6378135);}
            else
                rho=3.5e-12*exp(-(((r[1]-6378135)/1000)-380)/48);

            if (gravi_flag == 0)
                g = 3.9894065e14/(r[1]*r[1]);
            if (gravi_flag == 1)
                g = g_factor(L[1],l[1],r[1]);

            T=0;
            alpha=0;
            F_N = T*sin(alpha) + 0.5*rho*S*Cl*V[1]*V[1];
            F_T = T*cos(alpha) - 0.5*rho*S*Cd*V[1]*V[1];

/*calculations of changes in six variables*/
            dr = delta_r(V[1],gamma[1]);

            dV = delta_V(V[1],gamma[1],chi[1],
                r[1],L[1],g,m,omega,my,F_T);

            dl = delta_l(V[1],gamma[1],chi[1],r[1],L[1]);

            dgamma = delta_gamma(V[1],gamma[1],chi[1],
                r[1],L[1],g,F_N,m,omega,my);

            dL = delta_L(V[1],gamma[1],chi[1],r[1]);
```

```
    dchi = delta_chi(V[1],gamma[1],chi[1],
        r[1],L[1],g,m,omega,F_N,my);

    kr[q] = t * dr;
    kV[q] = t * dV;
    kl[q] = t * dl;
    kL[q] = t * dL;
    kchi[q] = t * dchi;
    kgamma[q] = t * dgamma;

    r[1]=r[0];
    V[1]=V[0];
    l[1]=l[0];
    L[1]=L[0];
    chi[1]=chi[0];
    gamma[1]=gamma[0];

    for (s=1;s<=q;s++)
        {
        r[1]=r[1] + kr[s] * a[q+1][s];
        V[1]=V[1] + kV[s] * a[q+1][s];
        l[1]=l[1] + kl[s] * a[q+1][s];
        L[1]=L[1] + kL[s] * a[q+1][s];
        chi[1]=chi[1] + kchi[s] * a[q+1][s];
        gamma[1]=gamma[1] + kgamma[s] * a[q+1][s];
        };
    };

ddr=ddV=ddl=ddL=ddgamma=ddchi=0;

for (q=1;q<=8;q++)
    {
    ddV = ddV + kV[q] * b[q];
    ddr = ddr + kr[q] * b[q];
    ddl = ddl + kl[q] * b[q];
    ddL = ddL + kL[q] * b[q];
    ddchi = ddchi + kchi[q] * b[q];
    ddgamma = ddgamma + kgamma[q] * b[q];
    };

r[1] = r[0] + ddr;
V[1] = V[0] + ddV;
l[1] = l[0] + ddl;
L[1] = L[0] + ddL;
chi[1] = chi[0] + ddchi;
gamma[1] = gamma[0] + ddgamma;
```

```
        r[0]=r[1];
        V[0]=V[1];
        l[0]=l[1];
/*reset of longitude if >360*/
        if (l[0]>2*PI)
            {
            l[0]=l[0]-2*PI;
            };
        L[0]=L[1];
        chi[0]=chi[1];
        gamma[0]=gamma[1];

        j++;
        }
        while (r[i]/1000-6378.149>=taem_altitude);

/*------------------------------------------------------------ */
/*-------------  End of Runge-Kutta algorithm --------------- */
/*------------------------------------------------------------ */

    fclose(taem);

    if ((L[0]/PI*180 < taem_latitude+5./7.) &&
        (L[0]/PI*180 > taem_latitude-5./7.) &&
        (l[0]/PI*180 < taem_longitude + 50./x) &&
        (l[0]/PI*180 > taem_longitude - 50./x))
        {
/*THE ACTUAL POSITION CORRESPONDS TO THE AIMED ENTRY INTERFACE */
        printf("\nExpected time to reach TAEM:   %f sec",time);
        return 1;
        }
    else
/*THE ENTRY INTERFACE CANNOT BE REACHED THIS TIME. GO ON.*/
        return 0;
}
```

D.3 The data bases

In the following an example of the format of the data bases containing the values of the magnetic index A_p and the 10.7 cm solar flux is given. Both data bases have the same format and are a mandatory part of the presented version of the NRLMSISE-00 atmosphere model and the related functions.

```
const double flux[54][366] ={

//1960//

{ 170,    172.1,  179,    189.8,  209.5,  211.4,  220.3,  215.2,
197.5,  190.6, 196.5,  180.8,  174.9,  172.9,  179.4,  179.6,
175.7,  172.8,  161,    154.1, 159,    168.7,  184.4,  205.9,
225.5,  237.3,  243.4,  247.4,  232.6,  225.8, 220.1,  220.9,
209.1,  211.1,  205.4,  205.4,  188.7,  184,    179.7,  179.7,
174.9,  172,    163.1,  164.3,  164.3,  157.2,  155.4,  150.5,
148.7,  143.5, 140,    153.8,  146.7,  141,    138,    145.1,
145.1,  145.2,  138.3,  138.4, 135.3,  135.5,  136.5,  137.6,
138.6,  133.8,  137.7,  139.7,  141.7,  130.9, 130.9,  128.1,
134,    133.2,  136,    141.1,  139.2,  132.3,  136.4,  142.4,
144.6,  149.4,  153.5,  157.5,  156.7,  160,    168.9,  175,
181,    193.2, 182.2,  201.2,  184.2,  179.4,  188.6,  182.5,
169.7,  165.7,  147.7,  148.7, 156.9,  160.1,  169.2,  180.4,
184.5,  191.5,  184.5,  179.6,  177.8,  171.7, 176.9,  164.8,
161.9,  168,    167.1,  148.9,  145,    142.1,  144.1,  155.5,
163.6,  154.6,  162.7,  160.8,  158.8,  154.9,  159,    165.2,
171.4,  173.4, 173.6,  183.8,  182.9,  173.7,  165.7,  165.6,
158.6,  154.5,  156.7,  156.7, 163.8,  168.1,  168.1,  167.2,
168.3,  167.2,  162.3,  170.5,  175.6,  174.8, 174.8,  163.5,
170.8,  171.8,  171.8,  177,    175.1,  180.2,  190.5,  190.5,
186.6,  183.5,  176.3,  172.2,  167.2,  171.3,  171.3,  162,
157.9,  143.6, 144.6,  137.4,  135.3,  134.3,  140.5,  136.4,
144.6,  160.3,  169.6,  190.3, 196.5,  200.6,  215.1,  214,
217.1,  219.2,  216.1,  206.8,  193.4,  182, 182,    171.6,
158.2,  146.7,  139.5,  143.6,  150.8,  148.8,  158,    164.2,
161.1,  157,    157.9,  152.7,  155.8,  164.1,  152.7,  153.8,
154.6,  153.6, 158.8,  150.5,  149.3,  144.2,  138,    128.7,
125.5,  129.7,  130.7,  137.9, 149.2,  156.4,  174.8,  192.2,
220,    240.3,  244.4,  246.5,  247.3,  253.4, 256.5,  239.9,
224.5,  205.8,  193.5,  174.9,  165.7,  161.6,  165.6,  153.3,
142.9,  131.7,  131.6,  134.6,  139.7,  155,    151.8,  144.7,
144.6,  151.7, 164.8,  172.9,  175.8,  177.8,  177.6,  179.7,
183.7,  183.7,  180.5,  179.5, 187.4,  192.3,  201.4,  197.1,
191.1,  186,    176.9,  163.6,  156.4,  149.3, 143.1,  133.1,
124.9,  121.8,  115.8,  112.7,  120.7,  132.7,  132.7,  132.5,
```

```
144.6,  143.6,  151.5,  159.5,  152.3,  159.3,  162.2,  166.2,
165.2,  165, 167,    153.8,  152.8,  148.7,  143.7,  140.7,
133.7,  128.6,  129.5,  131.5, 131.3,  121.4,  130.2,  127.2,
126.1,  123.4,  128.2,  129.2,  130.1,  143, 146.8,  155.7,
166.6,  173.5,  198.2,  186.3,  166.2,  178.2,  190,    181.3,
172.4,  162.3,  151.4,  148.3,  145.3,  137.4,  125.6,  114.7,
111.6,  109.6, 115.6,  117.4,  115.4,  117.4,  129.1,  134.3,
143.2,  150,    160.8,  156.9, 158.8,  149.8,  151.9,  148,
149,    141.9,  138,    134,    130.1,  136.1, 132,    123.2,
116.3,  113.3,  116.3,  114.3,  104.4,  101.4,  104.4,  109.2,
114.1,  122.9,  133.7,  142.6,  156.4,  160.3   },
```

D.4 Examples of the input files

D.4.1 orbit_input.txt

```
1000.00
400.00
279.16
51.6
1
1
1
////////////////////////////////////////////////////////////
1)  apogee      [km]
2)  perigee     [km]
3)  longitude   [deg]
4)  inclination [deg]
5)  latitude flag:
                        1=start at perigee minimum latitude
                        2=start at perigee maximum latitude
                        3=start at apogee minimum latitude
                        4=start at apogee maximum latitude
6)  direction flag:     1=eastwards
                        2=westwards
7)  deorbit flag:       0=no deorbit burn
                        1=deorbit burn to reach entry interface
                          specified in Deorbit.txt
```

D.4.2 manual_input.txt

```
101.783
7620.
-1.57
53.09
40.426
42.266
0
////////////////////////////////////////////////////////////
1) Initial Altitude             r[0]     [km]
2) Initial relative Velocity    V[0]     [m/s]
3) Initial Flight Path Angle    gamma[0] [deg]
4) Initial local Azimuth        chi[0]   [deg]
5) Initial Latitude             L[0]     [deg]
6) Initial Longitude            l[0]     [deg]
7) Deorbit flag        0=no deorbit burn
                       1=deorbit burn to reach the entry interface
                         specified in Deorbit.txt
```

D.4.3 launch_input.txt

```
5.24
307.22
40.22
0.
3
7100.
3

4.
1021000.
0.
3129.39
0.
118.344
156640.
118.344
19874.3

1.
990200.
0.
3129.39
0.
278.
90100.
278.
8165.

1.
298000.
0.
3521.79
280.
578.
25400.
578.
2026.

2700.
203.
203.

3500.
0.
118.344
```

```
700.
0.
278.
////////////////////////////////////////////////////////////
1)  Latitude                [deg]
2)  Longitude[deg]
3) launch azimuth      [deg]
4)  Lift-off time after ignition    [sec]
5) amount of stages[INTEGER]
6) payload mass [kg]
7)  additional mass flag [INTEGER]
(if there is any additional mass that is lost during the launch
(NOT the propellant, but e.g. water for cooling or fairing) this
flag contains the amount of mass items that are to be defined
further down!) e.g.
                    0 denotes NO additional mass
                    1 denots ONE additional mass item
and so on...

for EACH stage:
8)  number of engines   [FLOAT!!! e.g.: 2.0]
9)  thrust per engine  [N]
10)  angle between thrust and rocket axis (if any)  [deg]
11)  specific impulse [m/s]
12)  burn start [s]
13)  burn stop [s]
14)  propellant mass of stage (e.g. for ALL engines)    [kg]
15)  separation time [s]
16)  separation mass of (whole!) stage [kg]

after ALL stages have been characterized the additional mass has
to specified (if additional mass flag NOT 0):

17)  additional mass    [kg]
18)  start time of lost
19)  stop time of lost
```

D.4.4 Deorbit.txt

```
42.266
40.4313
101.78
-1.51
////////////////////////////////////////////////////////////
Entry interface longitude [deg]
Entry interface latitude [deg]
Entry interface altitude [km]
Entry interface FPA  [deg]
```

D.4.5 spacecraft_input.txt

```
2840.
3.8
0
0.
0
1.26
0.
0.
////////////////////////////////////////////////////////////
1)  mass [kg]
2)  reference surface [m^2]
3)  lift coefficient flag:
               0= take the following value
               1= call function to look up value from Cl_input
4)  lift coefficient Cl
5)  drag coefficient flag:
               0= take the following value
               1= call function to look up value from Cd_input
6)  drag coefficient Cd 7)  bank angle [deg]
8)  angle of attack [deg]
```

D.4.6 CD_input.txt, CL_input.txt

```
11 2
0.0 0.7 0.8 0.9 1.0 1.05 1.1 1.5 2.0 3.0 35.
0.0 180.0
2.22 2.22
2.22 2.22
2.17 2.17
2.46 2.46
3.09 3.09
3.2 3.2
3.08 3.08
2.4 2.4
```

```
1.92 1.92
1.28 1.28
1.28 1.28
////////////////////////////////////////////////////////////
1) # of MachNumbers,  # of AoA
2) Mach Numbers in ascending order!!!
3) AoA in ascending order!!!
4-...) Data
```

D.4.7 pitch_input.txt

```
0
14
0. 12. 90. 90.
12. 17.9 90. 86.3
17.9 20. 86.3 86.3
20. 36.8 86.3 76.8
36.8 61.3 76.8 60.
61.3 80.6 60. 46.3
80.6 100. 46.3 36.7
100. 117. 36.7 30.2
117. 120. 30.2 35.8
120. 205. 35.8 25.2
205. 319. 25.2 12.5
319. 400. 12.6 4.4
400. 450. 4.4 -1.
450. 567. -1. -12.6
////////////////////////////////////////////////////////////
1)  pitch\_flag  [INTEGER] 0=inertial pitch,
                            launch pad reference frame
                          1=local pitch, spacecraft reference frame
2)  amount of equations [INTEGER]
                           How many lines are about to follow?
3)  Data Format:        [FLOAT]

    start\_of\_time\_interval  stop\_of\_time\_interval
    pitch@start\_time      pitch@stop\_time

(The get\_theta function will use this data for linear
interpolation)
```

D.4.8 physical_model_input.txt

```
6378136.49
7.2921159e-5
3.98941e14
0.
350
2001
1
0
0.5
1
////////////////////////////////////////////////////////////
1)  Earth Radius                [m]
2)  Angular Velocity Earth      [rad/sec]
3)  Gravitational constant times Earth mass
4)  UT [sec]
5)  DOY             [DDD]  as integer
6)  YEAR            [YYYY] as integer
7)  atmospheric flag        0=no atmosphere
    (INTEGER)               1=analytical model
                            2=NRLMSISE-00 model
8)  gravitational field flag    0=spherical gravitational field
    (INTEGER)                   1=J-2 correction terms
9)  integration timestep        [sec]
10) printflag
    Defines which results are written in files:
                1 = every result is written in files
                2 = every second result is written
                3 = every third result is written
                ...
```

D.4.9 stop_condition_input.txt

```
1
11.7
////////////////////////////////////////////////////////////
1)  type flag:   (defines the type of stop condition)
    1=altitude at which the program is stopped coming down (e.g.: 120.)
    2=altitude at which the program is stopped for launch (e.g.: 80.)
    3=amount of orbits that are to be computed  (e.g.: 10)
    4=amount of seconds that the program is to run (e.g.: 1000.)

2)  limit:  contains the value of the stop condition
            Type depends on type flag: can be INTEGER or FLOAT
```

Erklärung:

Ich versichere, dass ich diese Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.

Heidelberg, den

Unterschrift: Sascha P. Quanz