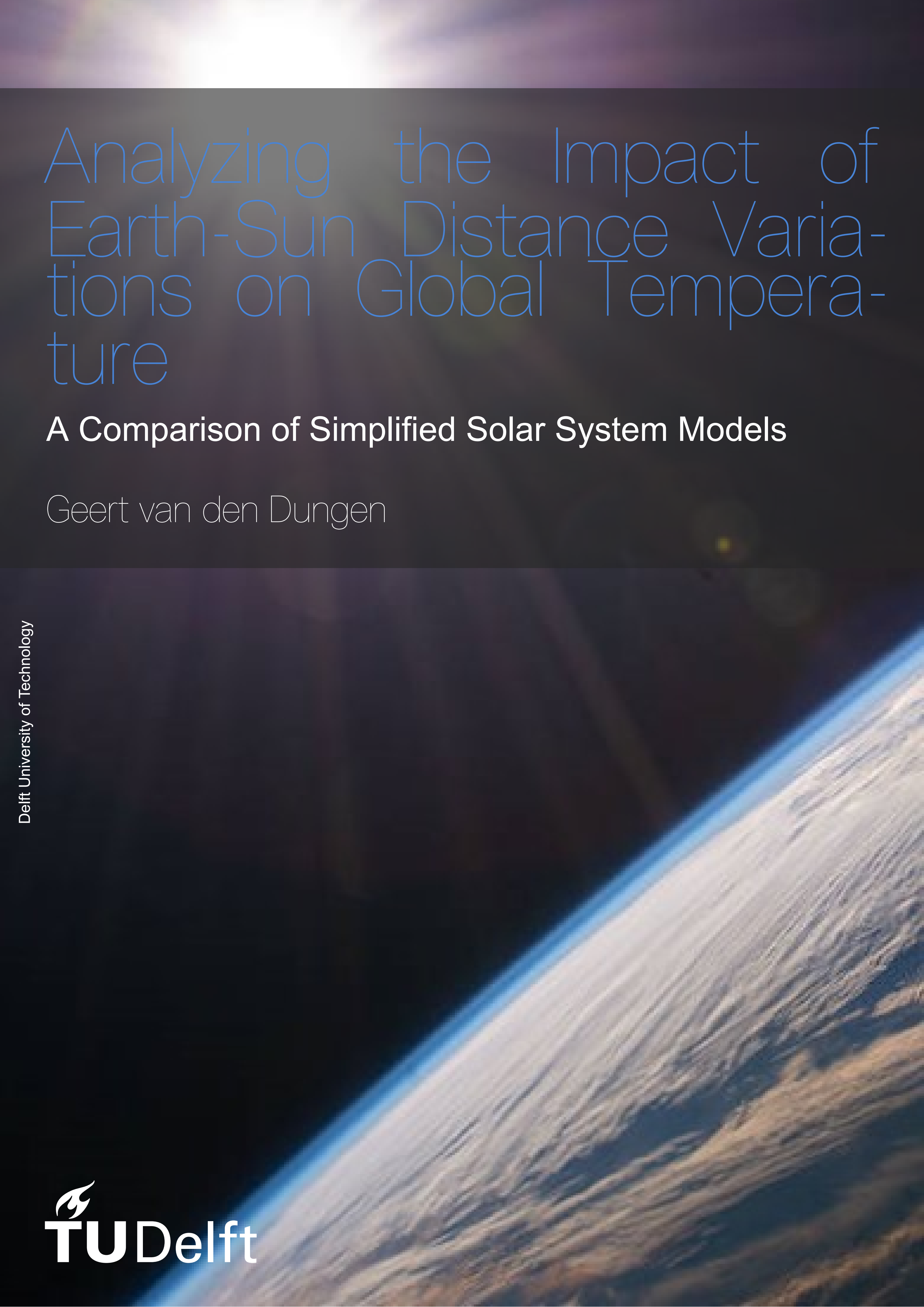
Analyzing the Impact of Earth-Sun Distance Variations on Global Temperature

A Comparison of Simplified Solar System Models

Geert van den Dungen



Delft University of Technology

TUDelft

Analyzing the Impact of Earth-Sun Distance Variations on Global Temperature

A Comparison of Simplified Solar System Models

by

Geert van den Dungen

In partial fulfillment for the degree of Master of Science
at the Delft University of Technology.

Student number:	4554566
Supervisor:	Marc Naeije

Cover: The sun shines above the Earth's horizon by NASA
Style: TU Delft Report Style, with modifications by Daan Zwaneveld

TU Delft

Preface

After a long period at the TU Delft, this thesis paper concludes my time at the university — an end to an important period in my life. I want to thank all the professors who taught there. Furthermore, my student society ”De Delftsche Zwervers” obviously does need to be mentioned here. Without the developments I've gone through, there would not be the current me.

Some people need to be specifically mentioned for their help. Professionally, Professor Marc Naeije and Professor Emeritus Ir. R. Noomen. for being very helpful supervisors during my literature study and thesis work. Besides that, Kevin, without you, I wouldn't have passed my Masters. Johan, for keeping me sane throughout not only my university but also before that, and definitely after too. And of course the Blub group, for all the support they are.

Geert van den Dungen
Delft, November 2024

Abstract

This thesis investigates how variations in the Earth-Sun distance influence global temperatures, by comparing a simplified model of the solar system with an existing paper from V.V. Zharkova, claiming that increasing temperatures can be explained naturally. Over a 5000-year period, numerical simulations including planetary gravitational influences, solar inertial motion, and Milankovitch cycles, this study looks at distance variations and Earth hemispheric differences in solar intensity due to albedo differences, to asses this statement. The result shows that while orbital mechanics influence the global temperature, Their role is minimal. It should see a slight decrease in temperature, and thus V.V. Zharkova's research does not represent the actual situation. This offers valuable insight into the relationship between the Earth's orbital mechanics and climate. However, further research into the accuracy of the model is required.

Contents

List of Figures

List of Tables

Nomenclature

Abbreviations

Abbreviation	Definition
AU	Astronomical Unit
BS	Bulirsch-Stoer
CPU	Central Processing Unit
CW	Chandler Wobble
JPL	Jet Propulsion Laboratory
MM	Maunder Minimum
NASA	National Aeronautics and Space Administration
txt	text (file saving denotation)
RK	Runge-Kutta(-Fehlberg)
RKDP	Runge-Kutta Dormand-Prince
SPICE	Spacecraft, Planet, Instrument, C-matrix, Events - system
SSB	Solar System Barycenter
TUDAT	TU Delft Astrodynamics Toolbox
UN	United Nations

Symbols

Symbol	Definition	Unit
$\bar{C}_{lm}$	Normalized coefficient of the Legendre function	-
D_r	Local distance to the Sun	m
D_{Earth}	Average distance between the Earth and the Sun	m
d	Apothem	m
H_{Earth}	Average solar intensity of the Earth	W/m^2
G	Universal gravity constant	6.6743E-11 $m^3kg^{-1}s^{-2}$
H_{local}	Local solar intensity	W/m^2
l	Degree; used in Spherical Harmonics	-
M_a	Mass of planetary body a	kg
m	Order; used in Spherical Harmonics	-
μ_a	Gravitational constant of planetary body a	ms^{-2}
$\bar{P}_{lm}$	Associated Legendre function of the first kind	-
R_a	Radius of planetary body a	m
$\mathbf{r}_{a,b}$	Position vector from planetary body b to planetary body a	m
$(\mathbf{r}_{Earth,SSB})_{model}$	Position vector from Earth to the SSB, taken from the model	m
$(\mathbf{r}_{Earth,SSB})_{JPL}$	Position vector from Earth to the SSB, taken from JPL	m
$\ddot{\mathbf{r}}_{a,b}$	Acceleration vector of mass a due to mass b	ms^{-2}

Symbol	Definition	Unit
$\bar{S}_{lm}$	Normalized coefficients of the Legendre function	-
T_{bb}	Black body temperature	K or C
U_a	Spherical harmonic potential of Body a	-
α	Albedo	-
α	Angle between the incoming solar rays and the Earth's rotation axis	rad
Δr_{norm}	Difference calculated as a norm	m
Δr_{magn}	Difference calculated as a magnitude	m
∇U_a	Inertial Potential Gradients for U_a	-
ϵ_{model}	Magnitude of the model error	-
θ	angle used by apothem calculations (Equation 3.8)	rad
σ	Stefan–Boltzmann constant	5.67E8 $Wm^{-2}K^{-4}$
♈	Vernal equinox	
$^\circ$	Degree (temperature or angle)	
″	Arc-seconds (used for angles	
#	Number	

1 Introduction

In recent decades, the issue of global warming has become increasingly evident and concerning. While the primary cause of this warming is widely debated, most agree that human activities are largely responsible. However, not all scientists share this view. In recent years, differing reasoning has been proposed: variations in the Sun's activity or the Earth-Sun distance [43]. This thesis will focus specifically on the Earth-Sun distance and the effect this has on our climate.

One of the most interesting and prevalent articles is from V.V. Zharkova [43], who claims that there is an increase in the total solar irradiance closely related to the solar inertial motion and the terrestrial temperature, indicating a further natural increase of 2.5 degrees Celsius by 2600 [43]. However, this claim is highly contested and should be extensively reviewed, since other peer reviews [37] seem to disagree on this topic highly. As such, it is important to critically examine these claims to either validate or refute them.

In the previously mentioned article, a 1,950-year (± 75 years) period is identified as correlating with several solar irradiance minima and maxima in the past [43], which in turn are linked to the current increase in Earth's temperature. Figure 1.1 presents the baseline magnetic field, represented by the magenta irradiance curve, which is used to support the argument for increased solar activity. However, this report will focus on the distance between the Earth and the Sun, using a model that also accounts for the influence of other astronomical bodies, such as planets and major moons. With that model, it is possible to either prove or reject the notion that solar motion has anything to do with the climate on Earth.

Another argument presented [43] suggests that it is not the overall change in solar intensity on Earth that affects the climate, but rather the variation in solar intensity between the Northern and Southern Hemispheres. Due to the higher reflectivity of water and ice compared to most land surfaces, land absorbs more solar radiation than water, especially when land is compared to snow and ice [16][17][33]. Since the Northern Hemisphere contains significantly more landmass than the Southern Hemisphere, seasonal changes in solar intensity could theoretically have a more pronounced warming effect. However, this hypothesis will need to be tested through modeling.

To answer all the unknowns previously mentioned, the following research question was proposed in my research plan:

How does the current perceived change in the distance between Earth and the Sun affect the global temperature?

To better answer this question, the following subquestions are presented:

1. Can the results as seen in V.V. Zharkova [43][1] be reproduced without changing the parameters of the solar system, with only a simplified model of solar variation?

[1]This article has been redacted, but is still contested by most authors and defended in further works. Therefore it is deemed relevant and still included in this report.

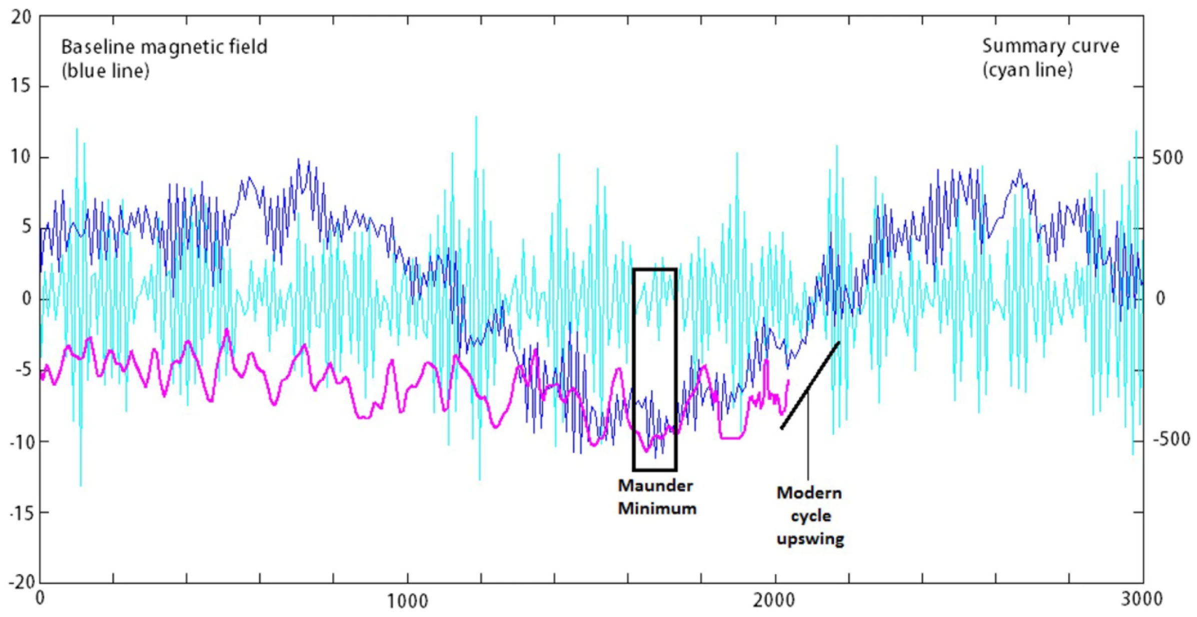


Figure 1.1: The close-up view of the oscillations of the baseline magnetic field (dark blue curve) from year 0 - 3000, with a minimum occurring during Maunder Minimum (MM). The irradiance curve (magenta line) is overplotted on the summary curve of the magnetic field (light blue curve). source: V.V. Zharkova [43].

2. What is the difference between the model used by this report and the model used by V.V. Zharkova?
3. Can the hemispheric effects (as described by V.V. Zharkova) be quantified in a simplified Earth hemispheric model?

In this report, Chapter 2, provides background information to contextualize the research within existing scientific knowledge. This includes a discussion of solar irradiance, the Earth-Sun orbital cycles (such as the Milankovitch cycles), and other planetary orbital variations. Subsequently, the methodology for the model and algorithm are discussed in Chapter 3, which outlines the methodology used to develop both the Earth-Sun distance model and the hemisphere model, along with the associated algorithms. Chapter 4 follows with a sensitivity analysis, as well as the verification and validation procedures. In Chapter 5, the results from the model are presented and discussed. After that, in Chapter 6, offers a conclusion based on the research findings, and finally, in Section 6.2, recommendations for future improvements and further research are provided.

2 Background

In this chapter, background information relevant to the analysis is provided. This information has been researched before the analysis and is presented here to contextualize the current research within the existing knowledge.

First, the distance between the Earth and the Sun concerning solar irradiance will be discussed in Section 2.1. Following that, the orbital variations between the Earth and the Sun will be analyzed in Section 2.2. Finally, variations in the orbits of other planets will be addressed in Section 2.3.

2.1. Solar Irradiance

The primary consequence discussed in this report regarding variations in orbital distance is their effect on solar irradiance and the temperature of the Earth. For this simplified analysis, the Earth's average temperature is calculated using a basic black body model. In this model, the Earth is represented as a "black body," a uniform object characterized by only a few parameters related to reflection, such as albedo. Additionally, this model does not account for any internal mechanisms or effects that influence temperature. This body simulation is done to simplify the calculations at this stage. Notably, the model omits surface irregularities — such as the differing reflectivity of water compared to land — and the natural greenhouse effects of the atmosphere. These effects have a very significant impact on the temperature model, since without these effects, the average temperature of the Earth is calculated to be -19.12 °C, while in actuality it is 15 °C globally [5].

In Figure 2.1, the relation between distance, solar irradiance, and black-body temperature can be observed. It can be seen that a small change in distance has a larger increase in both solar irradiance and temperature. This is due to the inverse square law of the solar irradiance and the black-body temperature formula used for the temperature. The equations used for the graph can be found in Equation 2.1 [31].

$$H_{local} = \frac{D_{Earth}^2}{D_r^2} \cdot H_{Earth} \qquad\qquad T_{bb} = (\frac{H_{local} \cdot (1-\alpha)}{4\sigma})^{\frac{1}{4}} \tag{2.1}$$

Here, H_{local} is the local solar intensity, H_{Earth} is the average solar intensity of the Earth, D_r is the local distance to the Sun, D_{Earth} is the average distance between the Earth and the Sun, T_{bb} is the black body temperature, α is the albedo of the planet, and σ is the Stefan–Boltzmann constant. The observed annual variance in the orbit of Earth (due to the eccentricity) is $5.0 \cdot 10^6$ km ($147.10 \cdot 10^6$ and $152.10 \cdot 10^6$ km)[5], which already translates to an almost 6% difference in solar irradiation and 4 degrees in temperature. Those degrees might seem unimportant, however, currently, the UN is stating that a 2-degree global temperature rise is catastrophic for the Earth [26]. Therefore, a distance variation in the order of a million kilometer could already be significant for the solar energy received on Earth and the corresponding black-body temperature.

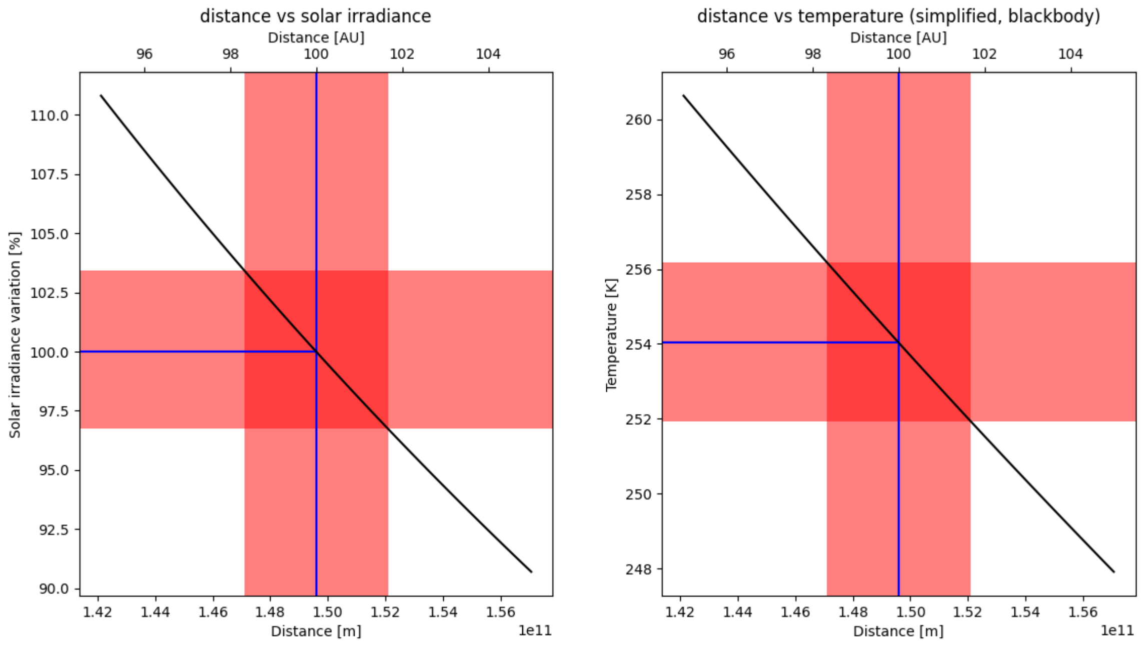


Figure 2.1: Effect of the Sun-Earth distance on the Solar Irradiance and the black-body temperature of the Earth, where the red area is the variance that occurs due to the eccentricity, the black line is the calculated data, and the blue line is the average position of Earth.

2.2. Earth and Sun orbit Variations

The Earth and the Sun are both attracting each other with gravitational forces and thus rotate around a shared center: the barycenter. The distance between the Earth and the Sun is therefore divided into two movements around this barycenter: the Sun's movement, the Solar inertial motion, and the Earth's rotation. In this section, first, the solar inertial motion will be discussed. Subsequently, the Earth's motion cycles will be described, going in order of their period from short to long periods. Finally, a conclusion for this section will be made.

2.2.1. Solar Inertial Motion

Due to the gravitational pull of the planets in the solar system, the center of the Sun is moving around the barycenter of the solar system. This movement, called solar inertial motion, means the distance between the Earth and the Sun also changes. The measured solar inertial motion can be found in Figure 2.2. Here, a 23-year variation is shown, between 1990 and 2013. It is an almost closing cycle around the barycenter of the solar system. However, if the actual percentage variance is calculated, it is as far as about a $13 \cdot 10^{-3} AU$ difference across from the barycenter, or a 1.3 % difference of the distance Earth-Sun. It is often approximated by a rose around the barycenter, where the cycle closes in this period [34], as can be seen in Figure 2.3.

However, the thermal inertia of Earth is very low [19], which means that Earth will react very slowly to changeable conditions, e.g. a change in the solar irradiance [34]. Moreover, this inertia is difficult to measure and will constantly change slightly due to changing conditions, such as atmospheric conditions, cloud coverage, ice coverage, water coverage, etc. Estimations of the effective heat capacity, which is the global heat capacity that sees an effect due to a perturbation of the climate with the length of the perturbation. This estimation is widely different in literature, from the 5-year time constant [34] to at least a factor of 3 more [15].

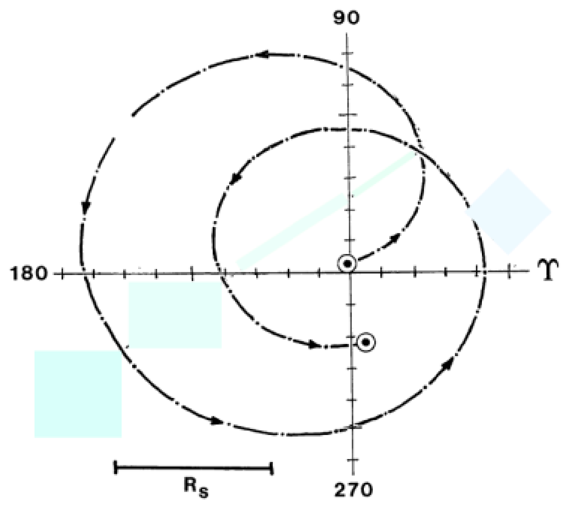


Figure 2.2: The orbit of the Sun around the barycenter of the solar system during the period 1990 to 2013. The barycenter of the solar system is at the origin and the points on the curve give the position of the center of the Sun at 200-day intervals. The plane of the figure parallels the equatorial plane of the Earth and the vernal equinox, defined as when the plane of the Earth's equator passes through the center of the Sun, is indicated on the abscissa. The units on the axes are AU x 10^{-3} or 1.496 x 10^{-8} m. Both figure and caption from Marsh 2020 [34].

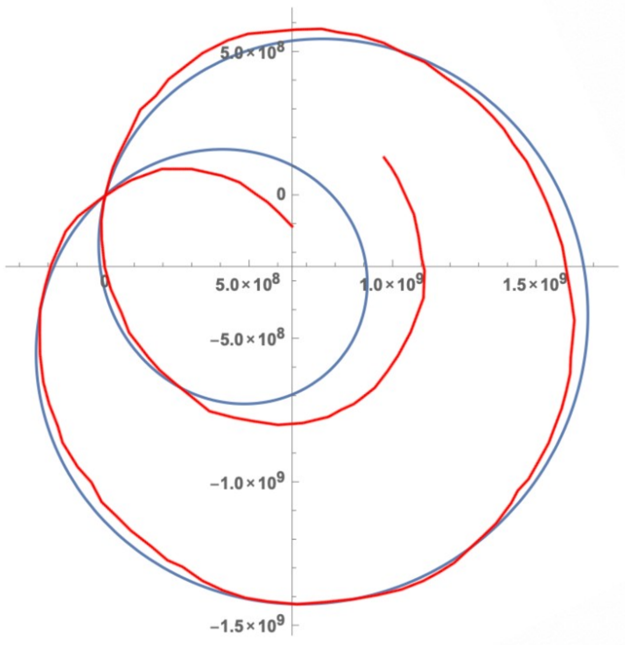


Figure 2.3: The 23-year variation around the sun (red) approximated by a rose around the barycenter (blue). The biggest error occurs around the endpoints of the measured Solar inertial motion. Figure from Marsh 2020 [34].

2.2.2. Earth-Sun Orbital cycles

By most predictions, Earth's orbit will be relatively stable over long periods [18]. However, that does not mean there are no variations in the orbit. The cycles of the Earth's and Sun's interactions have many differing lengths, from twice per day (such as tides) to a galactic year (smallest cycle (daily cycles) to the largest (Galactic year, 220-250 million years). Not all of these will be useful for the analysis. All cycles that are equal to or longer than a year will be discussed in this section, in order of their period, starting with the smallest. Furthermore, the relevant cycles for the analysis will be determined and discussed.

The cycles of the interactions between the Sun and Earth are divided into four categories: Calender cycles (less than a year), the Mid-range cycles (1 to 10,000 years), the Milankovitch Cycles (10,000 - 1,000,000 years), and the Galactic time cycles (more than 1,000,000 years) [23]. As mentioned previously, cycles less than a year will not be discussed. This is done because these cycles do not have any influences on the orbital motion, but are cycles of phenomena seen on Earth (such as (spring)tides, day/nighttime cycle, lunar month, and more.) Therefore, the Calender cycles will not be discussed.

Mid-range cycles cycles
Annual cycle
The Annual cycle has seen the most research out of all cycles since it is also very easily observable: it is responsible for the seasonal changes on Earth and by definition equal to exactly a year on Earth. The analysis of its amplitude can be found in Section 2.1, which comes to a 5% solar irradiance difference. Although this cycle is responsible for the most easily observable climate and temperature effects on Earth, it is too short of a cycle to be relevant for long-term climatic effects. It can, however, influence the global long-term climate due to the combination of the axial tilt of the Earth, and the difference in reflectivity of the Earth, as suggested by V.V. Zharkova [43]. This difference in reflectivity can be easily spotted on a globe or map. The surface of the Earth in the Southern Hemisphere has more ocean compared to the Northern Hemisphere. The difference in the surface has also a difference in reflectivity.

Chandler Wobble
The Chandler Wobble (CW) is one of the main components of motion of the Earth's rotation axis relative to the Earth's surface, also called Polar motion. It was discovered by Seth Carlo Chandler in 1891 [32]. The CW is one of the main eigenmodes of the Earth's rotation, related to the movement of the Earth's surface, with two periodic cycles of 428 days and 365 days, respectively [32]. It has no component related to the Sun or the Earth's orbital movement, and therefore should not be included in the analysis.

Solar Year
The solar year refers to the period in which Sunspot activity varies, which has a cycle of around 22 years [23][43]. The cycle is caused by a flip of the magnetic field of the Sun, creating a peak in solar activity, such as sun spots and coronal mass ejections. These spots change the luminosity of the sun, and therefore the intensity of the solar irradiance. This simplified model for the solar cycle will be used in the analysis. However, it will only be used in the second phase of the model, since it is not the focus of the research. The next solar maximum is predicted to occur in July 2025 [40]. However, the activity of the sun during the solar maxima is not very predictable: some cycles have a lower activity during their maxima than other cycles. The reason for this variability in maxima is currently unknown [35]. It is sometimes also known as the Hale cycle of 22 years, since after two maxima the magnetic field has flipped again and returned to the previous state of 22 years [23].

Milankovitch cycles
The Milankovitch cycles are changes in the orbital elements of the Earth orbit first found by M. Milankovitch [23]. They describe the long-term, 15,000 - 1,000,000 year, changes of several different factors, described in the sections below. In Figure 2.4, the Milankovitch cycle orbital effects are shown. It has been extensively researched previously that the Milankovitch cycles both on their own and all three together do not explain the current global warming [1][2][20][38].

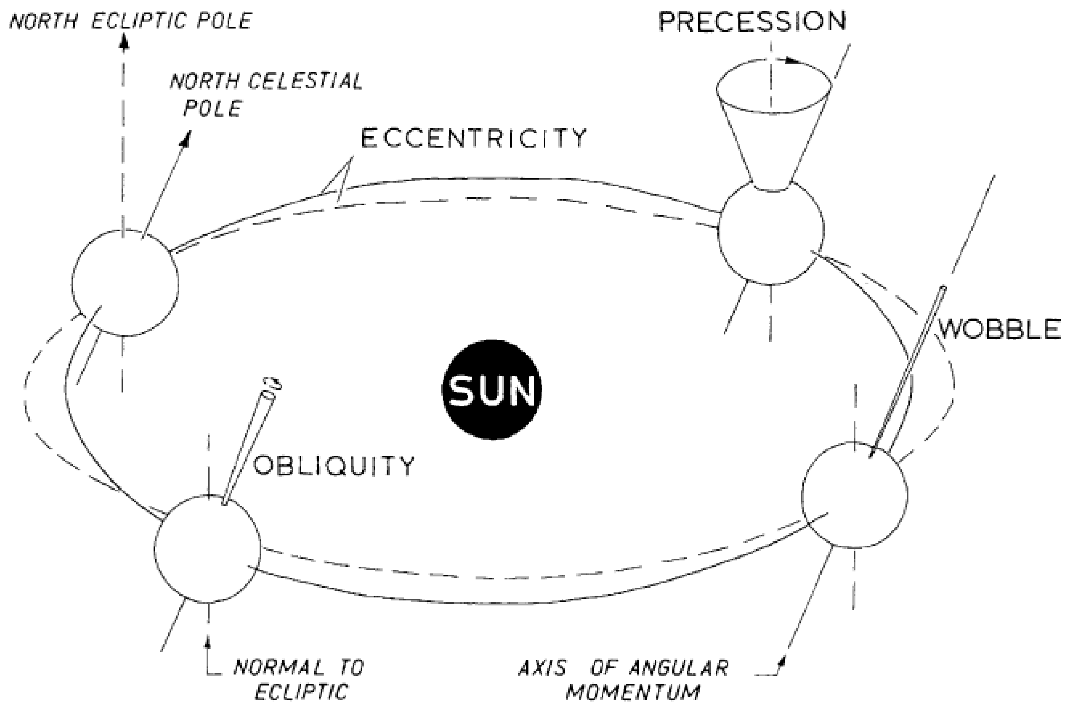


Figure 2.4: Orbital effects of the Milankovitch cycles and the Chandler Wobble. Figure from House (1995) [23].

Precession
Precession is a rotation of the Earth's axis, with a period of 25,771.5 years [1]. It is caused by the gravitational attraction of the Sun and Moon, and a very limited amount by the attraction of other planets [23]. It does not influence the distance between Earth and the Sun, however, due to the changing of the angle of the axis at certain points of the (slightly eccentric) orbit of Earth, it has an influence on the hemispheres where the solar intensity occurs and therefore should be included in the same manner as the annual cycle should be included for climate change when looking at which hemisphere of the planet is receiving the most solar radiation. Currently, perihelion (when the distance between Earth and the Sun is the lowest) is during winter in the Northern Hemisphere.

Obliquity
Obliquity is the tilt of the Earth's rotational axis. With a current tilt angle of 23.4°, but fluctuating between 22.1° and 24.5°. It has a period of about 41,000 years. It is confirmed to influence the climate, by changing the intensity of the solar irradiance by about 5% [23]. Currently, the tilt is decreasing slowly. It should result in warmer winters and cooler summers, therefore increasing the ice sheets at high latitudes and with the high reflectivity, decrease global temperatures [1]. Since it is only related to the Earth's rotational axis and not to the orbital path, it should be included in the same method and location as precession will be included.

Eccentricity
Due to the small eccentricity that the Earth currently has (0.0167 [5]), the distance between the Earth and the Sun changes during a year between 147.1 and 152.1 million km, a variation of 3.4%. This means that during perihelion, the Earth receives about 6.7% more solar energy than during aphelion [1]. However, this eccentricity is not constant over time, it changes with several cycles that combine in a cycle of around 100.000 years [28]. It varies between 0.0034 and 0.0580. The current eccentricity is slowly decreasing and approaching its most circular phase of the cycle. It is caused by the pull of mainly the two largest gas giants, Saturn and Jupiter. Perihelion occurs around the 3rd of January each year and aphelion around the 4th of July [1]. Since eccentricity is a critical element of the orbital parameters, it needs to be included in every analysis and model about the distance between Earth and

the Sun. Subsequently, the difference in eccentricity should be included in the model, although due to the time scales, there should not be a large effect on the global temperature.

Galactic time cycles
Galactic Year
The period to rotate around the center of the Milky Way is around 220-250 million years. It is thought to influence major glaciation cycles, but periods do not match the timings found. There is no influence currently found between the galactic year and either the climate on Earth or the orbit of Earth [36]. Additionally, the period of the galactic year is so large that it can not have such an influence that is being analyzed here. Therefore, the effect of the galactic year will be fully ignored in the proposed analysis.

2.2.3. Conclusion

In Section 2.2.1, the solar inertial motion was discussed. Here, it was seen that the Sun moves around the barycenter of the solar system. This movement can be approximated by a rose, with a maximum orbital distance change to the Earth of 1.3%. However, the assumption will introduce an error in the movement, and will therefore need to be studied, therefore this assumption will not be made in the model used for this analysis. Subsequently, there are numerous Earth-Sun cycles discussed in Section 2.2.2. Of the discussed cycles, the annual cycle and the eccentricity Milankovitch cycle will need to be included in the base model. Furthermore, the precession and obliquity Milankovitch cycles have to be included in the hemisphere model. Finally, in the simple Sun model, the solar year cycle must be included. The Chandler wobble and the galactic year will not be included in any model performed by the analysis performed in this report.

2.3. Planet orbit Variations

In this section, the movement of each of the 7 planets other than Earth will be briefly described in sections 2.3.1-2.3.7. Although Pluto, the Asteroid, and the Kuiper belts all have a gravitational attraction to all objects in the solar system, their influence is assumed to be negligible. For each planet, only the important parameters related to its orbit and the attraction forces to Earth will be discussed, including the forces of their moons (if they exist). The order of the planets will be from closest to furthest from the Sun. The ephemeris and other planetary data can be easily found in several databases, such as JPL[1] and TUDAT[2]. Therefore, only the most important data parameters will be mentioned here. After each planet is discussed, the possibility of a planet X will be investigated. Subsequently, a table with standard parameters for all (found) planets is presented in Section 2.3.9, as well as a graph of orbital accelerations on Earth. Finally, a conclusion is given in Section 2.3.10.

2.3.1. Mercury

Mercury has the most eccentric orbit of all planets in the solar system, with an eccentricity of 0.2056. Furthermore, It has an orbital period (year) of 87.969 Earth days [8], and it has no moons. Due to this high eccentricity, the distance to the Sun will differ between 0.3075 and 0.4667 AU (46-70 million km). Furthermore, it has an orbital resonance with the Sun, such that a day on mercury lasts exactly 2 mercury years. Its orbit is inclined by 7° from the solar system plane, the largest of all in the solar system [8]. Simulations have shown that Mercury has a possibility (1%) to have an unstable long-term orbit due to the influence of the perihelion of Jupiter. This unstable orbit can, according to the performed situation, result in situations, where in 5 billion years, Mercury falls into the Sun. Another possibility is that it can disrupt the inner solar system and Mercury, Venus, Earth, or Mars could collide [27][30].

[1] https://ssd.jpl.nasa.gov/horizons/
[2] https://docs.tudat.space/en/latest/index.html

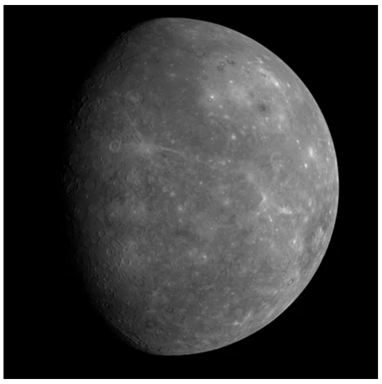


Figure 2.5: Mercury (NASA image database)

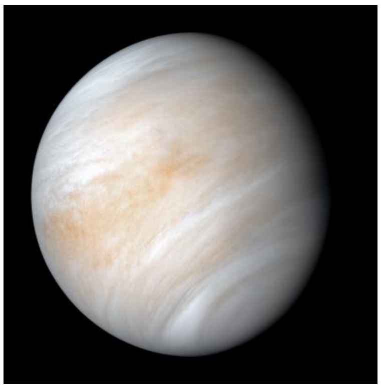


Figure 2.6: Venus (NASA image database)

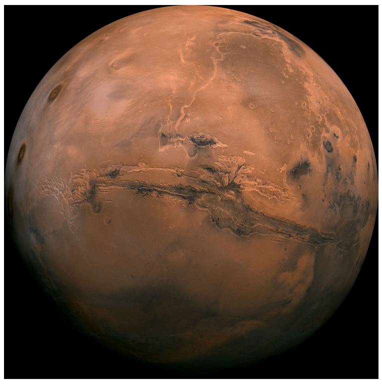


Figure 2.7: Mars (NASA image database)

Mercury also has a precession. It was proposed by Einstein as a test of general relativity [13]. It is now observed to be 574.10″±0.65 (arcseconds) per century [3] by the MESSENGER spacecraft.

2.3.2. Venus

Venus has an extremely stable and circular orbit compared to other natural bodies. It has an eccentricity of 0.0068 and a distance to the Sun of 0.7184-0.7282 AU (108.21-108.94 million km). It also has an orbital period of 224.70 Earth days [11]. It has no moons, theorized to be because of either the strong solar tides [39] or tidal deceleration caused by Venus itself [42]. It has an inclination to the solar plane of 3.395°. The precession is measured to be 8.6247″±0.0005 per century [3]. It has no other orbital parameters or significant influences not previously mentioned.

2.3.3. Mars

Mars has a similarly stable orbit to Venus, however, the influence of Jupiter and the asteroid belt are significantly more predominant. Due to the complexity of the asteroid belt modeling, this will not be included in the analysis. It has a rather large eccentricity of 0.0934 and a distance to the Sun of 1.405 to 1.639 AU (206.65-249.26 million km) [7]. The orbital period is 686.98 Earth days and an inclination to the solar plane of 1.848° [7]. Mars has similar cyclic variations as Earth. However, the period for most of these variations is millions of years, so these effects can be assumed to be negligible [29]. The orbital eccentricity cycle of Mars is 96,000 Earth years, very similar to Earth. The precession is measured to be 1.351″±0.001 per century [3]. Finally, Mars has two moons: Phobos and Deimos. However, these moons are extremely small and insignificant for gravitational influences [7].

2.3.4. Jupiter

Jupiter is one of the most interesting planets for the analysis. It has relatively one of the biggest impacts on Earth's orbit due to its mass. It has a distance from the Sun of 5.2 AU (778 million km) and an orbital period of 4332.589 Earth days (11.86 years). The eccentricity is 0.0487° [6]. The inclination to the solar plane is 1.304°. Jupiter has 95 confirmed moons, of which 4 are massive enough to include in the analysis given in this report [6]: Io, Europa, Ganymede, and Callisto. Their masses are prevalent enough to include in a more detailed analysis model if time permits.

Jupiter has been known to clean the solar system of debris [21]. It has such a large influence on the orbits of the solar system that most planets are closer to Jupiter's orbital plane than that of the sun, except for Mercury. Jupiter is responsible for astronomic phenomena such as the Kirkwood gaps [21]. These Kirkwood gaps are regularly spaced voids in the asteroid belt, where significantly fewer asteroids are present. These asteroids are, over a long period, removed by Jupiter's gravitational

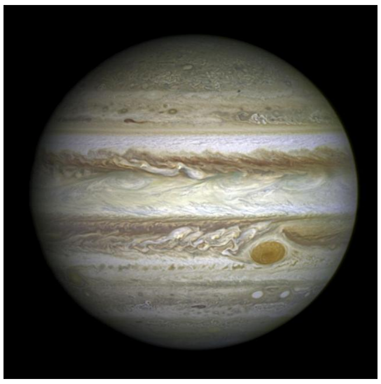


Figure 2.8: Jupiter (NASA image database)

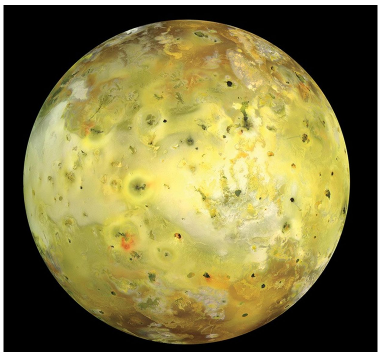


Figure 2.9: Io (NASA image database)

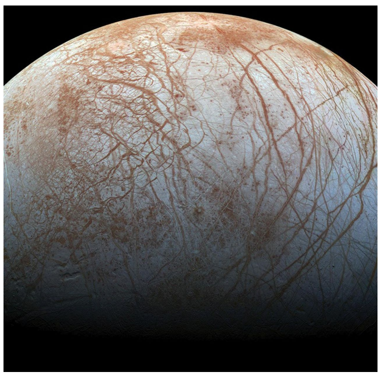


Figure 2.10: Europa (NASA image database)

Figure 2.11: Ganymede (NASA image database)

Figure 2.12: Callisto (NASA image database)

attraction. Furthermore, Jupiter is responsible for the asteroid belt that exists in the first place: due to its gravity, any significantly larger object that currently exists will be either fragmented or attracted by Jupiter and removed from the solar system. Finally, Jupiter has created two swarms in its orbit: the Trojan asteroids. These are asteroid swarms that are either in front of or behind Jupiter's orbit. Whether Jupiter protects the inner solar system or creates more asteroids or meteorites is not yet definitively known [21]. It is known that Jupiter's obliquity is not caused by Uranus [12].

2.3.5. Saturn

Saturn is the second largest planet in the solar system, thus having a large influence on planetary orbits in the solar system. It has an eccentricity of 0.052 and a period of 10,759.22 Earth days (about 29.5 years). The distance to the Sun is 9.195 and 9.957 AU (1.3-1.5 Billion km). Its inclination to the solar plane is 2.48° [10].

Saturn has 146 confirmed moons and of course the well-known rings. Although other planets also have rings, Saturn has the most pronounced. However, all but one of them are too small to have an orbital influence on any object in this analysis. The only exception is Titan, which could be included together with the 4 Galilean moons of Jupiter. Saturn has an interesting rotation, comprised of separate 3 periods. This report will not go into detail about these rotations, because these rotations do not influence the orbital parameters of the solar system. Also interesting is that Saturn does not have any Trojan asteroids, while Mars, Jupiter, Uranus, and Neptune all have confirmed Trojan asteroids. Orbital resonance is believed to be the cause of their absence [22]. Finally, There is some evidence that leans towards Saturn being in a spin-orbit resonance with Neptune, where the axial precession of Saturn matches Neptune's orbit [4].

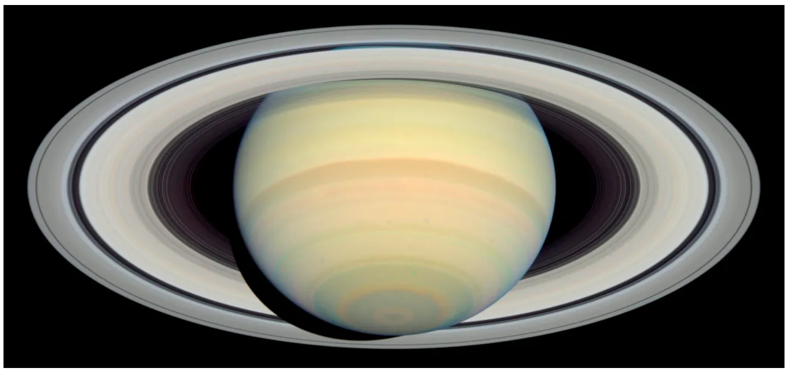


Figure 2.13: Saturn (NASA image database)

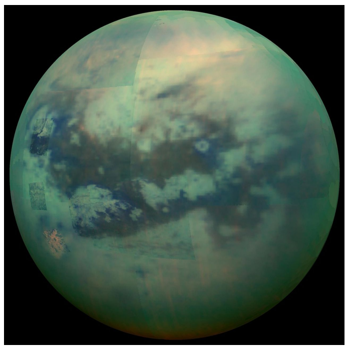


Figure 2.14: Titan (NASA image database)

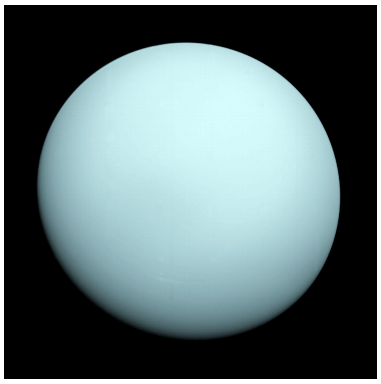


Figure 2.15: Uranus (NASA image database)

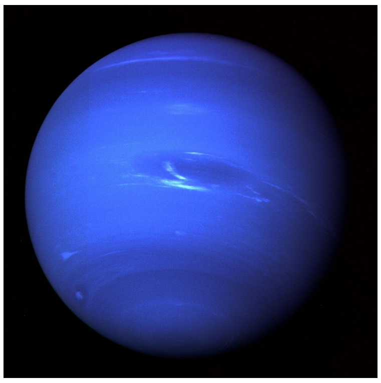


Figure 2.16: Neptune (NASA image database)

2.3.6. Uranus

Uranus has a relatively stable orbit. The period is 30,685.4 days (84 years) and has an eccentricity of 0.0469. Due to the fact it only has done around 2.9 orbits since its discovery, a lot of the orbit is still unknown with observations, although a lot of simulations have revealed details of the orbit. The distance to the Sun is 18.2-20.0 AU (2,732.696 - 3,001.390 million km), and the orbital inclination to the solar plane is 1.770°. Uranus has 27 confirmed moons, none of which are large enough to influence the solar system. The orbital discrepancies of the observed orbit of Uranus have been modeled and later led to the discovery of Neptune.

2.3.7. Neptune

Neptune has a very low eccentricity of only 0.00867. However, due to its large distance to the Sun, this still means that the distance to the sun varies between 29.81 and 30.33 AU (about 4.5 billion km). It has an orbital period of 164.79±0.1 years and thus has only completed its first orbit since its discovery in 2011. The orbital inclination between the orbit of Neptune and the solar plane is 1.77° [9]. Neptune has a large influence on the region behind it in the solar system, such as the Kuiper belt. Due to both the difficulty in modeling such a belt, and the negligible effect this belt will have, it will not be modeled in the analysis. Neptune has a total of 14 moons, and similar to Uranus, they are all so small that there will be no gravitational effects from them [9].

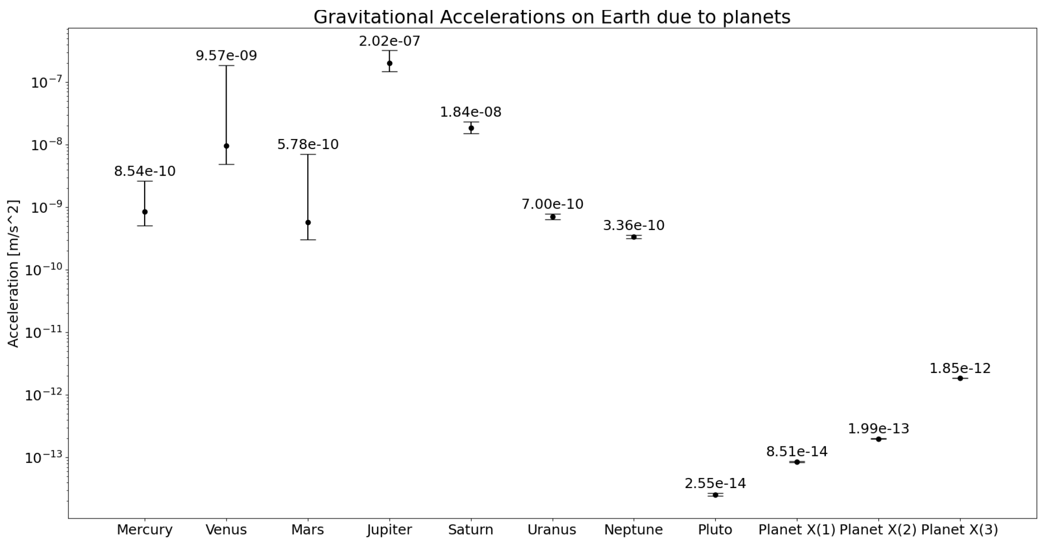


Figure 2.17: Gravitational acceleration of the planets in our solar system on Earth, including Pluto and the three options for Planet X, discussed in this chapter. Shown here are the gravitational accelerations on Earth by each object in their most remote state (on opposite sites of the Sun), their closest state (aligned with the Sun), and when there is a 90° difference between the locations in the respective orbit (the dot). This last situation is also the numerical value shown. For this calculation, all orbits are assumed circular and completely aligned in the same solar plane.

2.3.8. Planet X

One of the questions at the beginning of the investigation was the influence of planet X on the orbits of our solar system. Although this topic has been extensively researched in the past, several options for a planet X are still open and theoretically possible, especially when looking at a very large semi-major axis. For this subsection, only L. Iorio (2011) [25] will be used since this paper is a comprehensive summary of the search for Planet X.

Constraints for the search for the planet are taken by only using simulated masses of Mars or greater. Doing such research, there are three possibilities found for the combination of mass and distance to the Sun: a Mars-massed planet at 150-200 AU, a 70% of Earth's mass planet at 250-450 AU, and a 4 Jupiter masses planet at 3500-4500 AU. The possibilities of the presence of planet X are low, but not impossible.

In Figure 2.17, the gravitational acceleration of Earth due to the gravitational forces of the planets was calculated. Here, all 7 planets are included, together with Pluto and the three possibilities of Planet X found in Iorio (2011) [25]. Shown are the gravitational accelerations on Earth by each object in their most remote state (on opposite sites of the Sun), their closest state (aligned with the Sun), and when there is a 90° difference between the locations in the respective orbit (the dot). This last situation is also the numerical value shown. For this calculation, all orbits are assumed circular and completely aligned in the same solar plane. As can be seen, the gravitational acceleration is very small compared to the other planets, however, it is larger than Pluto. Therefore, for the analysis performed, it could be an interesting factor to be included. However, the stability of the solar system as a whole could be impacted by such a choice, so implementation should be done carefully and with proper verification and validation. This means that the model after inclusion needs to be thoroughly checked for instabilities and other discrepancies that prove that Planet X is implemented correctly. If the solar system model including Planet X is distinctly further from validated data due to this inclusion, then Planet X is not implemented correctly. However, it was decided that planet X is not included into the model, due to time constraints.

2.3.9. Planet detail table

Metric [Unit]	Mercury	Venus	Earth	Mars
Mass [10^24 kg]	0.33	4.87	5.97	0.642
Average distance to Sun [AU]	0.3871	0.7233	1.000	1.522
Orbital Period [Earth days]	87.969	224.70	365.256	686.98
Eccentricity [-]	0.2056	0.0068	0.017	0.0934
Inclination to Solar plane [°]	7.0	3.395	0.000	1.848
Precession [″ per century]	574.10±0.65	8.6247±0.0005	-	1.351±0.001
Gravitationally interesting Moons (total moons) [#]	0 (0)	0 (0)	1 (1)	0 (2)

	Jupiter	Saturn	Uranus	Neptune
Mass [10^24 kg]	1898	568	86.8	102
Average distance to Sun [AU]	5.2	9.576	19.1	30.07
Orbital Period [Earth days]	4,332.589	10,759.22	30,685.4	60,189±36
Eccentricity [-]	0.0487	0.052	0.047	0.00867
Inclination to Solar plane [°]	1.304	2.48	0.770	1.770
Precession [″ per century]	-	-	-	-
Gravitationally interesting Moons (total moons) [#]	4 (95)	1 (146)	0 (27)	0 (14)

	Io	Europa	Ganymede	Callisto
Parent body	Jupiter	Jupiter	Jupiter	Jupiter
Mass [10^21 kg]	89.3	48.0	148.2	107.6
Average distance to parent body [10^6 km]	422	671	1070	1883
Orbital Period [Earth days]	1.8	3.6	7.2	16.7
Eccentricity [-]	0.004	0.009	0.001	0.007
Inclination to Solar plane [°]	0.04	0.47	0.18	0.19

	Moon	Titan
Parent body	Earth	Saturn
Mass [10^21 kg]	73.5	134.6
Average distance to parent body [10^6 km]	384	1222
Orbital Period [Earth days]	27.3	15.9
Eccentricity [-]	0.055	0.029
Inclination to ecliptic[°]	5.1	0.33

Table 2.1: Planet data sheet

2.3.10. Conclusion

All planets are important for the solar system model, however, the orbits still need to be known. The more in-depth variables (such as the ephemeris) of each planet can be easily found in databases such as TUDAT (SPICE) and JPL, so they are not included in this chapter. Several planets have moons that are massive enough to be able to influence the solar system, which should be included in a more in-depth model. The inclusion is decribed in Section 3.1. For the simple model, including all the planets and having a stable system is already a significant task. Their orbital plane is mostly aligned, with only Mercury as an outlier. Jupiter and Saturn are the most influential planets to be modeled for the solar system dynamics. In regards to the asteroid belt, the Kuiper belt, and the Oort cloud, these should not be modeled in the analysis performed due to both difficulty in easy modeling, and their negligible influence on the outcome. Finally, Planet X could theoretically be included in a very detailed step of the model. It is bordering on the influence it has on the solar system dynamics if it even exists. Therefore, It will be not in the analysis performed in this report.

3 Methodology

In this chapter, the methodology of the research is discussed. At first, a basic model of point masses is proposed in Section 3.1, which will be further developed in subsequent sections. This section will also discuss the period to be analyzed by the model and the bodies to be used in the model. In Section 3.2, a new function for periodic averaging will be introduced, the aim of which is to decrease the computational time to run the model. Due to the inclusion of this to-be-created function, the specific type of propagator and integrator for the model cannot be determined in advance. Therefore, the analysis of the integrator and propagator combinations to be used will be addressed later in Section 3.3, along with their settings. However, the third major component typically used in TUDAT analysis software, the optimizer, will not be necessary, as there will be no orbit that requires optimization; it only needs to approximate as closely as possible the actual configuration of the solar system. There are more perturbation forces (such as solar radiation pressure) acting in the solar system, but these are not taken into account.

In Section 3.4, the more complex calculation formulas for spherical harmonics will be analyzed to determine whether they are necessary for the desired level of detail or simply an unnecessary increase in computational resources. Subsequently, in Section 3.6, an additional model analyzing the differences in Sunlight intensity between the northern and Southern Hemispheres will be discussed. Finally, in Section 3.7, a conclusion for this chapter will be presented.

3.1. Model basis, time period and bodies used

As a starting point for the model, the TUDAT[1] software package will be utilized to simulate the orbits. This package is specially designed and maintained to help TU Delft students model solar system activities, such as planetary movement, but also spacecraft orbits. TUDAT allows us to leverage the foundational components provided by itself, rather than developing them from scratch. The software includes features like the ephemerides of the planets sourced from the SPICE database[2], which can also be cross-referenced to the database of JPL [14]. The use of the JPL database will be further discussed in Chapter 4.

The TUDAT package will model the solar system based on accelerations, where all astronomical bodies will rotate around the solar system barycenter. The formula for simple point mass accelerations is provided in Equation 3.1.

$$\ddot{\mathbf{r}}_{a,b} = -\frac{GM_a}{\left|\mathbf{r}_{a,b}^3\right|} \cdot \mathbf{r}_{a,b} = -\frac{\mu_a}{\left|\mathbf{r}_{a,b}^3\right|} \cdot \mathbf{r}_{a,b} \tag{3.1}$$

Here, the acceleration performed by mass b onto mass a is shown. In this equation, $\ddot{\mathbf{r}}_{a,b}$ represents the acceleration vector, G is the universal gravity constant, M_a is the mass of planetary body a, μ_a is the gravitational constant of planetary body a, and $\mathbf{r}_{a,b}$ is the distance between planetary bodies a and

[1]https://docs.tudat.space/en/latest/index.html

[2]https://naif.jpl.nasa.gov/naif/data.html

b. More complex formulas, such as those involving spherical harmonics, can also be applied for cases requiring greater accuracy. This will be discussed further in Section 3.4.

A 5000-year period has been selected for my analysis, as a shorter duration would diminish observable effects and may not reveal a cycle, potentially resulting in an unstable system. A significantly longer period would include the same results as the 5000-year analysis (since it would still include the same 5000 years); however, if certain effects weren't shown in the 5000-year model, then these effects should not be a result of the natural causes analyzed by this research, since their period an amplitude combination would not be able to explain the effects. Moreover, extending the analysis period would considerably increase the model's computational time, requiring more resources (e.g., time) without necessarily yielding additional insights. Therefore, the model will analyze a period of 2500 years both into the future and the past. To simplify the dates used, the midpoint of the analysis is set to January 1, 2000, at 00:00. Since this represents only a 25-year difference, it is not expected to significantly affect the outcomes, as this is a negligible time frame compared to the overall 5000 years of analysis. Additionally, climatic and temperature changes occur gradually, making it highly unlikely that 25 years would produce a noticeable impact.

In Section 2.3, the orbital variations of the planets have been discussed. Here, the decision needs to be made whether to include which moons of the solar system. For this, a simple analysis is run. Due to this analysis, a 50% decrease of error was observed when included all six moons chosen in the background analysis, on an analysis period of 100 years. Therefore, the six moons will all be included in the final model. The results were $1.45 \cdot 10^7 km$ without moons and $6.5 \cdot 10^6 km$ including moons.

3.2. Periodic averaging function

Since the model will analyze 2500 years into both the future and the past, with 2000 as center point, it should not propagate the entire orbits of all celestial bodies. A new function was proposed to quicken the analysis and reduce the computational time for the full 5000-year period. This function takes the effects of a single orbit and uses that outcome to simulate several subsequent orbits, thereby decreasing the simulation time needed by eliminating the time-consuming time steps necessary for these following orbits. However, due to the absence of resonance in the dynamics of the solar system, the application of this orbit has to be done with caution, and of course should be verified and validated before the inclusion into the model. This function is therefore difficult to predict on its detailed method since extensive testing is necessary before it can be used with any confidence in the result. This was therefore an important part of the model proposed at the start.

During the initial implementation of the described function, a significant challenge arose, which was spotted in advance: the periods of the planets do not align, caused by a lack of resonance. This absence of resonance complicates the correct averaging for all planets. After numerous tests of different methods, each focused on 1 year - a baseline year - to estimate a following 1-year period - the resulting year. The results for the Earth were within a reasonable boundary since the difference could be simply added; the Earth was in the same spot in its orbit. For the other planets, a Keplerian approach was employed to project the position of the planets further along their current trajectory based on the baseline year. This method proved effective for both planets and moons with shorter orbital periods than a year and planets with a longer period than a year, although it yielded greater errors for those with longer orbital periods.

However, both methods proved insufficient when estimating the Solar Inertial Motion, as discussed in Section 2.2.1. Although the Sun's motion around the solar system's barycenter is semi-regular and could potentially be estimated using a rose model, it was decided not to employ this approach. Significant discussions have arisen regarding the assumptions made by Zharkova in her work, and it was decided to fully simulate the motion of the Sun, using TUDAT software and the function described here. Unfortunately, it was found that the Solar Inertial Motion cannot be accurately estimated for a year without complete simulation, which undermines the purpose of the proposed function.

As a result, the model must fully simulate the Solar Inertial Motion, the model also has to fully simulate the accelerations acting on the Sun, due to which all planets also need to be fully simulated. Preliminary calculations using the time taken by the model, established with an initial integrator and propagator (classical Runge-Kutta method with 4th order coefficients and Cowell, respectively), suggested that based on a simulation run spanning only 25 years, a full simulation run of 2500 years - one side of the analysis period - would take approximately 10 to 15 minutes. This runtime estimation for the full model, along with the anticipated challenges and time required to develop a function for simulating the Solar Inertial Motion, was sufficient to decide to not work further on the function and to fully simulate the solar system for the 5000 years of the analysis in the model.

3.3. Integrator and Propagator Selection

After it was decided to not implement the periodic averaging function, discussed in the previous section, the selection for integrator and propagator could be done. The function could impact the workings of the integrator and the propagator, so the selection was delayed until the function was completed. However, now that the function is deprecated, the selection can be performed.

TUDAT solves the formulas for planetary movement with the help of a propagator and integrator combination. The integrator solves the differential equation formulas, and then the propagator advances them to the next step. The size of the steps depends on the settings.

Integrator

The methods used by the integrators can be divided into two methods: fixed time step or variable time step with tolerances. The fixed-time step only needs the step size as a variable, while the variable step size includes both minimum and maximum step sizes, with tolerances determining the actual step size. The integrator uses these tolerances to adjust the step size, which is particularly beneficial for spacecraft approaching planetary objects. These variable step sizes allow for the algorithm to better make use of the resources and time of the computer.

To compare the available integrators, a baseline must be established. For this, the JPL Horizons database has been used. Further discussion about this database can be found in Chapter 4. Specifically, the value measured for error is the magnitude of the error in the Earth's position vector relative to the solar system barycenter, as defined in Equation 3.2.

$$\epsilon_{model} = ||(\mathbf{r}_{Earth,SSB})_{model} - (\mathbf{r}_{Earth,SSB})_{JPL}|| \tag{3.2}$$

In this equation, $\mathbf{r}_{Earth,SSB}$ represents the position vector of Earth from the solar system barycenter (SSB). The term $(\mathbf{r}_{Earth,SSB})_{model}$ refers to the position vector calculated by the model, and $(\mathbf{r}_{Earth,SSB})_{JPL}$ refers to the position vector received from the JPL database. Finally, ϵ_{model} represents the magnitude of the model error.

The integrators are evaluated based on two metrics: accuracy and CPU time, which is the time required to complete the analysis. Three types of integrators are considered: the Runge-Kutta method, the Bulirsch-Stoer method, and the Adams-Bashforth-Moulton method. Since the results will be compared to the JPL database, it was chosen to only use the fixed time step functions, as these align with the JPL database. If variable time steps would be used, the JPL time step could fall in between the variable time steps, leading to an interpolated comparison and thus introducing additional errors. Consequently, the variable time step functions are not assessed. This excludes the Adams-Bashforth-Moulton method, as well as the variable time step versions of the Runge-Kutta and Bulirsch-Stoer methods. Therefore, the analysis focuses solely on the fixed-time step variants of the Runge-Kutta and Bulirsch-Stoer methods. The specific variants chosen for analysis include the 4th order Runge-Kutta (RK4), the 5th order Runge-Kutta-Fehlberg with an embedded 4th order (RK45), the 6th order Runge-Kutta-Fehlberg with an embedded 5th order (RK56), the 8th order Runge-Kutta-Fehlberg with an embedded 7th order (RK78), the 7th order Runge-Kutta Dormand-Prince with an embedded 8th order (RKDP87), and three Bulirsch-Stoer methods with 4, 6, and 8 substeps (BS4, BS6, and BS8).

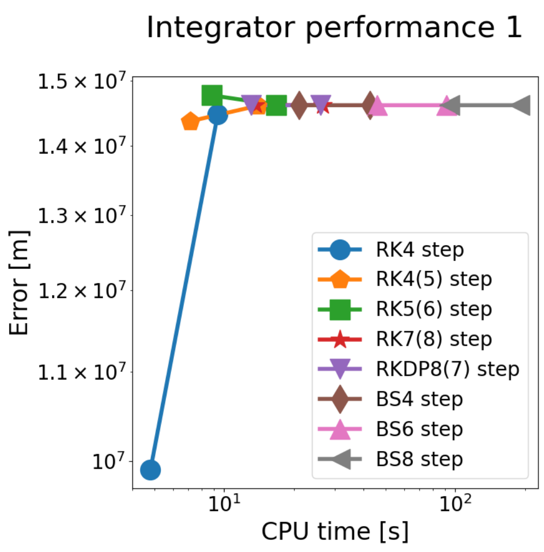


Figure 3.1: First step in integrator selection. Shown here are 8 integrators, compared in error with relation to the JPL database and in CPU time taken.

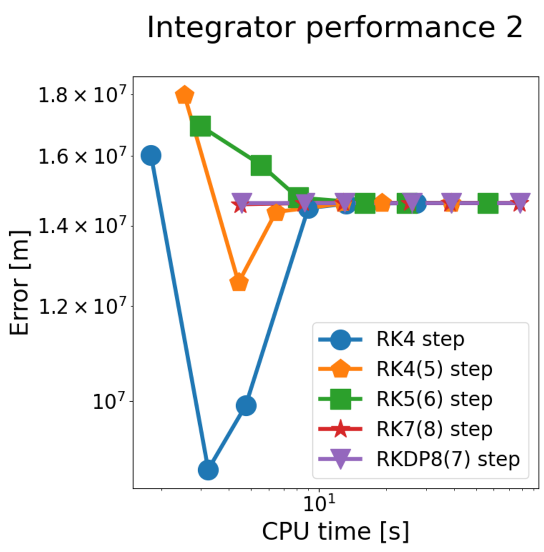


Figure 3.2: Second step in integrator selection. Shown here are 5 integrators, compared in error with relation to the JPL database and in CPU time taken.

Running a full 2,500-year simulation for all integrators would be too time-consuming; therefore, the selected integrators are evaluated over a shorter 100-year run, for only two step sizes. After this analysis, more detailed step sizes will be examined for the remaining integrators. The chosen step sizes are designed to be divisible by a Julian year (365.25 days), which simplifies calculations by eliminating the need to account for leap days. The applicable step sizes are 1 hour, 2 hours, 3 hours, 6 hours, 9 hours, and 18 hours. For the initial analysis, 9-hour and 18-hour steps are selected. Figure 3.1 presents the results of this first analysis. Here, it is clear that for the time taken, the three Bulirsch-Stoer methods all require significantly more time than the Runge-Kutta methods, without achieving a substantial reduction in error. Therefore, in the next step of the integrator selection, only the six Runge-Kutta methods remain.

In the second step of integrator selection, all the time step settings are considered and simulated with a 100-year-long simulation. These results can be found in Figure 3.2. This graph shows that all methods converge to a ’line of error convergence’ in the model as the time step decreases. This indicates that the error is driven by one of the other components and not the integrator under these settings. Since the goal is to pick the integrator with the best combination of the lowest CPU time and minimal error, initially it would seem that the RK4 method would be ideal, since it has by far the lowest error and at the same settings very low CPU time. However, with a lower degree of step size, the error increases to the same error as other integrators.

This increase in error for lower step sizes can be explained by an inaccurate modeling error that coincidentally creates a positive effect on the error, thereby reducing it. With this in mind, the RK4 method only has a comparable error as the other methods with a step size that is 2 degrees smaller, which also results in significantly increased CPU time. Taking this analysis in mind, Based on this analysis, two integrators — RK78 and RKDP87 — are nearly indistinguishable from one another, both lying on or very close to the previously mentioned ’line of error convergence’. For these two methods, a new graph (Figure 3.3) is created to highlight their differences. In this analysis, the 6 previously discussed moons (Section 3.1) are included as point masses, in addition to the settings used in the previous step. Here, the same effect is seen as in Figure 3.2; the integrators converge to a ’line of error convergence’. However, it can be observed that the RKDP87 converges more slowly to this line, requiring a smaller step size and consequently higher CPU time. Therefore, the RK78 is selected as the integrator for the model. In Figure 3.3, this integrator and step size combination is indicated by a red point.

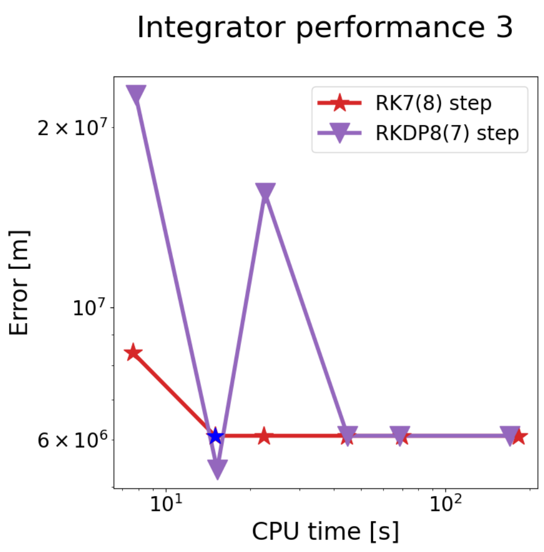


Figure 3.3: Last step in integrator selection. Shown here are the last 2 integrators, compared in error with relation to the JPL database and in CPU time taken. The moons are influencing the result, the period is 100 years. The blue star indicates the integrator and setting combination chosen.

Propagator

The same method as the integrator was planned for selecting the propagator. However, due to the use of the solar system barycenter, the Encke and Gauss Modified Equinoctial propagators do not work, since no central body was used in the algorithm. Subsequently, when using the Gauss Keplerian or one of the three Unified state model propagators (quaternions, modified Rodrigues parameters, and exponential map), the algorithm also failed, because the algorithm 'couldn't remove the central point gravity'. Therefore, only the Cowell propagator worked within the algorithm, and therefore, this propagator was chosen as the propagator to be used by the model.

3.4. Spherical harmonics

As mentioned in Section 3.1, to improve the accuracy of the model of the solar system, the point mass acceleration formula can be replaced by spherical harmonic formulas. These equations can be found in Equation 3.3 and Equation 3.4. The spherical harmonics are a more detailed modeling of the gravity of a planetary body. Since the mass of a planet is not uniformly distributed on most astronomical bodies - exceptions are for example stars, which are so massive that their gravitational pull more effectively distributes the mass evenly - their gravity field is also not uniform. With the use of the spherical harmonic formulas, you can model these discrepancies in the gravitational field of any given astronomical body with the use of constants.

$$\ddot{\mathbf{r}}_{a,b} = \mathbf{a}_{a,b} = \nabla U_a(\mathbf{r}_{a,b}) \tag{3.3}$$

$$U_{\hat{a}}(\mathbf{r}_{a,b}) = \frac{\mu_a}{r_{a,b}} \sum_{l=1}^{\infty} \sum_{m=0}^{l} (\frac{R_a}{\mathbf{r}_{a,b}})^l \bar{P}_{lm}(sin\phi_{a,b})((\bar{C}_{lm})_a cos(m\theta_{a,b}) + (\bar{S}_{lm})_a sin(m\theta_{a,b})) \tag{3.4}$$

Here, the symbols used are: ∇U_a are the inertial potential gradients at position $\mathbf{r}_{a,b}$ for spherical harmonic potential U_a of Body a, $\bar{P}_{lm}$ is the associated Legendre function of the first kind, $\bar{C}_{lm}$, and $\bar{S}_{lm}$ are normalized coefficients of the Legendre function, related to degree l & order m, and R_a is the radius of planetary body a.

Not all planets have a high level of detail in their known spherical harmonic constants. What is known and included in the TUDAT software is shown in Table 3.1. However, the detail required for

Planetary Body	Available Parameters	Source
Earth	Full gravity field, up to degree and order 200	GOCO05c via GFZ
Moon	Full gravity field, up to degree and order 200	gggrx1200 via PDS
Mars	Full gravity field, up to degree and order 120	jgmro120d via PDS
Venus	Full gravity field, up to degree and order 180	shgj180u via PDS
Mercury	Full gravity field, up to degree and order 160	jgmess160a via PDS
Jupiter	Zonal coefficients up to degree 8	Nature paper [24]
Galilean Moons (Io, Europa, Ganymede, Callisto)	μ, C_{20}, C_{22}	IMCCE ephemerides

Table 3.1: Available Spherical Harmonics parameters in the SPICE module. Source: [41]

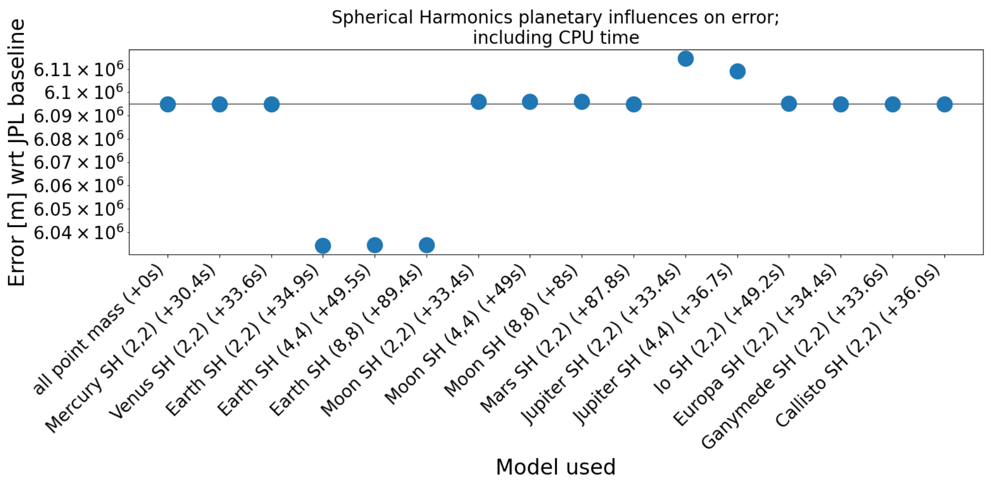


Figure 3.4: Spherical Harmonics selection step 1. base time was 14.6s

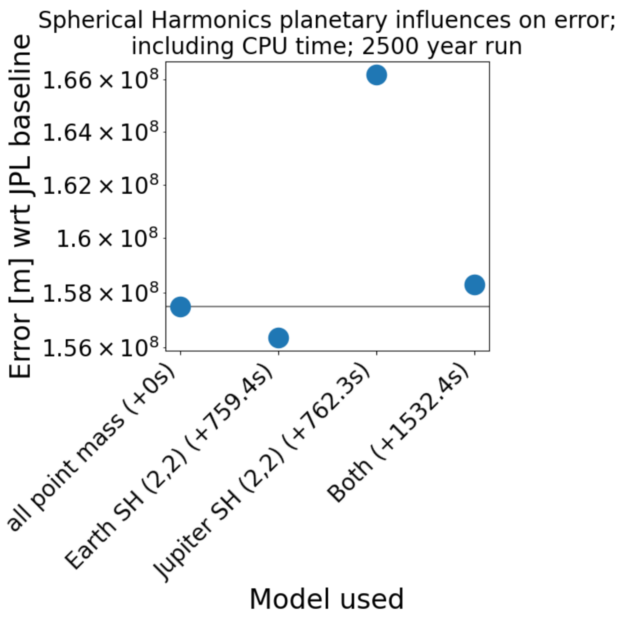


Figure 3.5: Spherical Harmonics selection step 2. base time was 665.5s

spherical harmonics decreases quickly with distance. Therefore, the current level of known spherical harmonic constants should be sufficient for the analysis. Although the formulas above are essential for the calculations, the TUDAT software only needs to be configured as a spherical harmonics model. If the correct constants are provided, it will automatically handle the formulas presented.

To decide whether spherical harmonics were necessary for the model, an analysis was run. In this analysis, the available spherical harmonics were activated one at a time to determine their influence on both time and accuracy. For this analysis, the time simulated was again 100 years, and the value for error is the maximum error concerning the JPL database, similar to the method for integrator selection. The result can be seen in Figure 3.4. Here, it can be observed that for the majority of planets or moons, the spherical harmonics do not have a significantly different influence over the point mass, however, as can be seen in the labels below, the time taken does increase significantly. The only significant difference can be spotted when the spherical harmonics are used for the Earth and Jupiter. It can also be seen that increasing the order and degree of the Spherical harmonic does not result in a significantly different error above the degree and order of 2, while still significantly increasing CPU time taken.

For a more detailed analysis, a 2500-year run was performed with the spherical harmonics of Earth and Jupiter; a run without any spherical harmonics, a run with only Earth's spherical harmonics up to degree and order 2, a run with only Jupiter's spherical harmonics up to degree and order 2, and a run with both Earth and Jupiter spherical harmonics up to degree and order 2. The result of this analysis can be found in Figure 3.5. Here, the error does change significantly, but it never decreases significantly. However, the CPU time taken does increase more than two for the cases where 1 Spherical harmonic is used, and for the case where both are used, the CPU time has more than tripled. Due to this large increase in CPU time, while not offering a significant decrease in positional error, it was chosen to not include any spherical harmonics in the model.

3.5. Result calculation

The magnitude of the changing distance between the Earth and the Sun can be calculated in two different methods: the norm and the magnitude difference. These equations can be found in Equation 3.5 for the norm and Equation 3.6 for the magnitude.

$$\Delta r_{norm} = ||\mathbf{r}_{Earth} - \mathbf{r}_{Sun}|| \tag{3.5}$$

$$\Delta r_{magn} = ||\mathbf{r}_{Earth}|| - ||\mathbf{r}_{Sun}|| \tag{3.6}$$

The parameters used are the difference calculated as a norm (Δr_{norm}), the distance calculated as a magnitude (Δr_{magn}), the distance between Earth and the Solar System Barycenter ($\mathbf{r}_{Earth}$), and the distance between the Sun and the Solar System Barycenter ($\mathbf{r}_{Sun}$). The norm gives a better image of the absolute distance between the Sun and the Earth, while the magnitude is mostly important for comparison regarding validation. There, the magnitude will give a better representation of a changing distance between objects, while the norm will give a better positional error.

3.6. Hemisphere model

After the initial research question has been answered by the method described earlier, another research question remains: 3. Can the hemispheric effects (as described by V.V. Zharkova et al.) be quantified in a simplified Earth hemispheric model? To answer this question, the solar intensity on both Earth Hemispheres has to be modeled. The reason this can affect climate is the albedo or reflectivity of both Hemispheres. The Northern Hemisphere has comparatively less land area compared to the Southern Hemisphere, while the Southern Hemisphere has much more sea area. Additionally, the ice sheet distribution - on land and sea combined - is also not evenly distributed. The areas - land, sea, and ice - all have a different reflectivity. Ice and snow have a much higher reflectivity than land, and will therefore reflect much of the energy of the Sun back to space, instead of absorbing it and warming the Earth. So, if the Northern Hemisphere sees an increase in solar intensity during the northern summer months, this could have the potential to increase the global temperature. Since modeling the albedo or reflectivity itself has a large component of uncertainty, since it involves estimating land, sea, and ice coverage, it was decided to not estimate the albedo of the Hemispheres, however, to only model the solar intensity on the Hemispheres throughout the period analyzed.

To model the Hemispheres, the percentage area of the Hemispheres in Sunlight is not really relevant. What is relevant, is the percentage of Sunlight that falls on each Hemisphere. The difference is observed from the angle of incidence. If the angle is 90°, then the solar intensity per square meter is the highest. Since the solar intensity per meter is not important, but the solar intensity per Hemisphere is, the percentage of Sunlight per Hemisphere is more useful.

For this, the three-dimensional case can be simplified to a two-dimensional problem; The Sunlight that falls on the Earth is in the shape of a disk. Therefore, the formula just needs to calculate the area of the circle that corresponds to the Southern Hemisphere, and the inversion ($100\% - Southern\%$) is then the Northern Hemisphere. This is known as the formula for a circular segment, shown in Equation 3.7 and with Figure 3.6.

$$A = \frac{R^2}{2}(\theta - sin\theta) \tag{3.7}$$

$$d = Rcos(\frac{\theta}{2}) \rightarrow \theta = 2cos^{-1}(\frac{d}{R}) \tag{3.8}$$

$$d = Rcos\alpha \tag{3.9}$$

$$\theta = 2cos^{-1}(\frac{Rcos\alpha}{R}) \rightarrow \theta = 2cos^{-1}(cos\alpha) \rightarrow \theta = 2\alpha \tag{3.10}$$

Since the southern area is calculated from the angle between the incoming solar rays and the Earth's rotation axis, denoted as α, then θ can be determined with geometry and Equation 3.8, where the equation for d, the apothem, is shown. α can be determined with right angles - since the Sun rays are always perpendicular to the disk, representing the Earth - and is shown in Equation 3.9, also linked to the apothem. Then, both formulas for apothem (Equation 3.8 and Equation 3.9) are combined in Equation 3.10, such that a link between the green area in Figure 3.6 and α has been found. These formulas can then be used to calculate the percentage of Sunlight that falls on the southern and Northern Hemispheres.

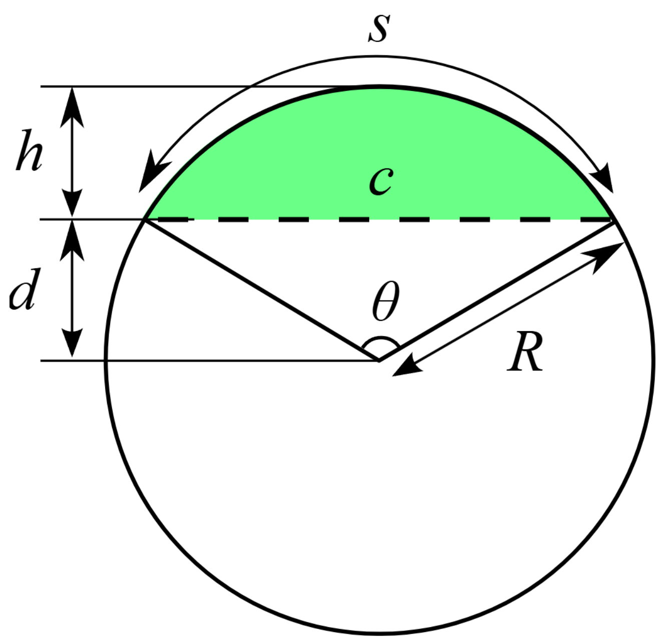


Figure 3.6: Circular section diagram, used for calculating the Hemisphere percentage. Marked in green is the area which is to be calculated. It corresponds to the Sunlight that falls on the Southern Hemisphere. The entire disk represents the Sunlight falling on the Earth.

After the percentage of Sunlight that falls on the Southern and Northern Hemispheres has been calculated, the solar intensity for each Hemisphere can be calculated. This is done by using the inverse square law, using the distances calculated described in the previous sections. With this distance, the changing solar intensity can be obtained, and together with the percentages calculated as described above, the solar intensity change can be found. Then, the statements of V.V. Zharkova [43] can be tested using this solar intensity change.

3.7. Conclusion

In this chapter, the methodology for the algorithm and model were discussed. First, the period was set at 2500 Julian years (365.25 days each) in the future and past, centered around the 1st of January 2000. The Sun and all 8 planets are included, rotating around the solar system barycenter, such that the solar inertial motion can be observed, together with our Moon, the 4 Galilean moons (Io, Europa, Callisto, Ganymede), and Saturn's moon Titan. Then, the averaging function was found to be ineffective and thus scrapped. Subsequently, the integrator and propagator could be decided as the RK7(8) integrator and Cowell propagator. When testing for Spherical Harmonics, the very slight error decrease in one of the models did not hold up against the more than twofold increase in CPU time, so only point masses are included in my model. Finally, a hemispheric model was discussed, where the solar intensity per Hemisphere on Earth can be calculated and, together with the distance calculated before, the influence of the Hemispheres in combination with solar intensity will be tested against the statements of V.V. Zharkova, which will be done in the next chapter.

As a final note regarding the methodology, the most important code files are available in Appendix A. Due to size limitations, the rest of the code files (verification, validation, plotting) used for this algorithm are available to be analyzed and reused by request.

4
Sensitivity, Verification & Validation

In this chapter, Section 4.1 covers the verification and validation processes. This section outlines the steps taken to ensure the accuracy and reliability of the model, comparing its outputs against established benchmarks and real-world data to confirm its validity. Following this, the sensitivity analysis is conducted and discussed. This is presented in Section 4.2, where we explore how variations in input parameters affect the model's results and assess the model's robustness to different initial states.

4.1. Verification and Validation

During the algorithm's implementation, verification and validation methods were applied from the outset to ensure the reliability of the model. Every time a new function was implemented that influenced the data handling of the model - such as the splitting of the runs into multiple parts due to memory issues, or the saving of data to a txt file between runs, such that plotting becomes more time-efficient - it was verified that these changes did not affect the results. Additionally, the Python 'assert' function was used in several code segments to halt the analysis if certain conditions were not met. For example, after reading and processing data from the model and the JPL database, the epoch of the runs was compared to the epochs of the JPL database to ensure that the data remained consistent with the original epoch. This check ensured that no unintended changes occurred during processing.

Plots were also generated to verify that the planets behaved as expected. One example of this was when an error was detected in the velocity of Mars. The velocity was incorrectly passed through the code and was found to be 1000 times higher than the actual value, resulting in an escape trajectory. This error was identified early through the plot shown in Figure 4.1

As mentioned earlier, the JPL Horizons database served as both a baseline for selecting the integrator, propagator, and spherical harmonics, and as a validation tool for the model. The uncertainties in the JPL database vary by planet, typically ranging from 0.5 to 30 km for terrestrial planets, and up to 4000 km for gas giants [14]. The period for which this uncertainty is given is between the years 1800 - 2200. While the 4000 km margin for the gas giants might seem large, it is a small fraction of the total distance, which spans millions of kilometers. As such, these uncertainties do not significantly impact the accuracy of the model's validation.

In addition to its role in selecting model parameters, the JPL Horizons database was also used to assess any significant differences between the model and the database. Figure 4.2 presents an analysis where the JPL database was extended beyond the scope of this study, allowing for a comparison of the overall shape and the identification of any cycles. Early indications of periodicity can be seen, particularly in the norm plot, which seems to approach the midpoint of an upward curve, and in the Z-coordinate, which clearly shows the changing acceleration in the position curve.

Another question raised was whether the results would differ significantly when starting the model from the middle in both directions. To address this, the conditions from the backward run (at -500 years, or 500 BC) were used to initialize the model, and the simulation was run for a full 5000 years. The difference between the two runs is shown in Figure 4.3. While the results do change, the magnitude of

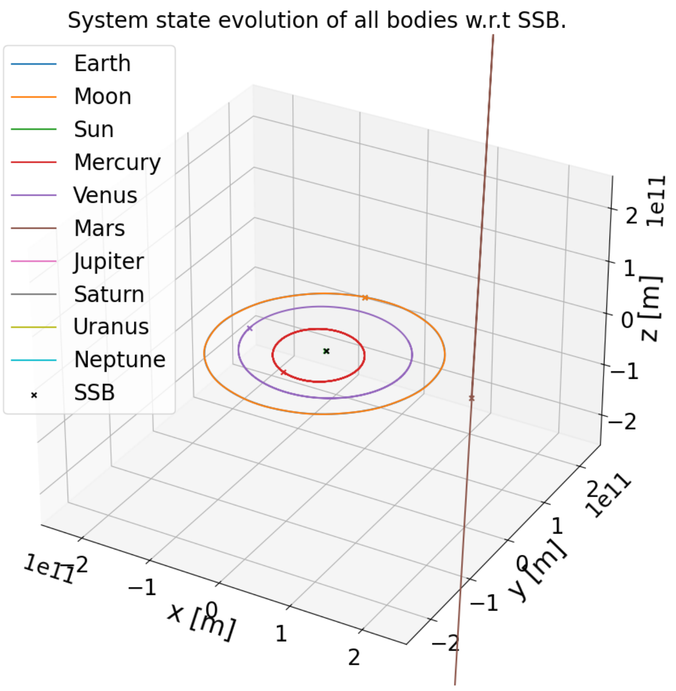


Figure 4.1: Failed fun example caught by verification. As shown here, Mars did not want to remain in the solar system. The error was found to be a multiplication of the velocity to 1000 times the actual value.

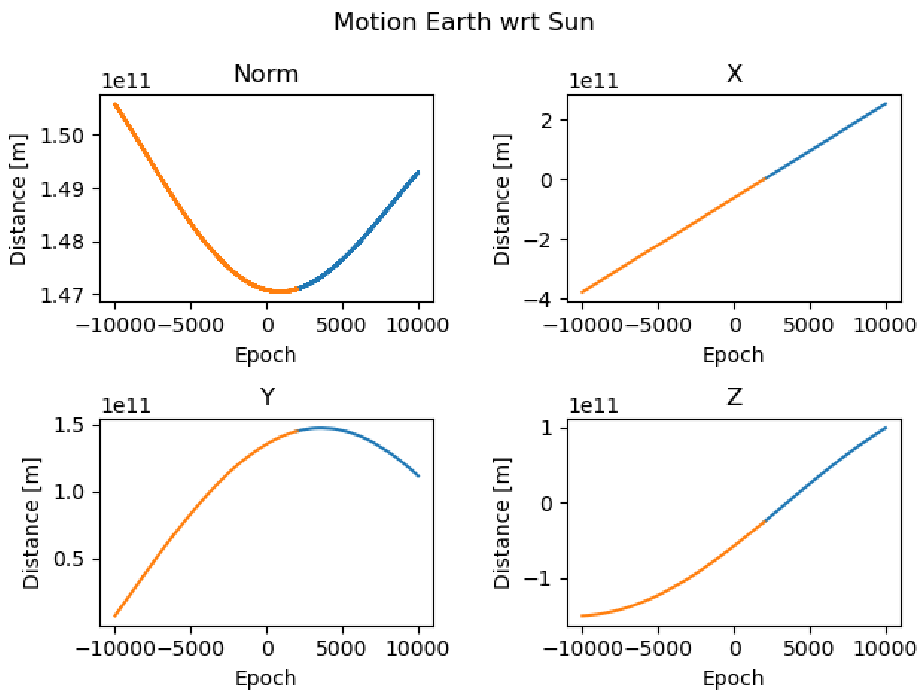


Figure 4.2: JPL data plotted for 20,000 years

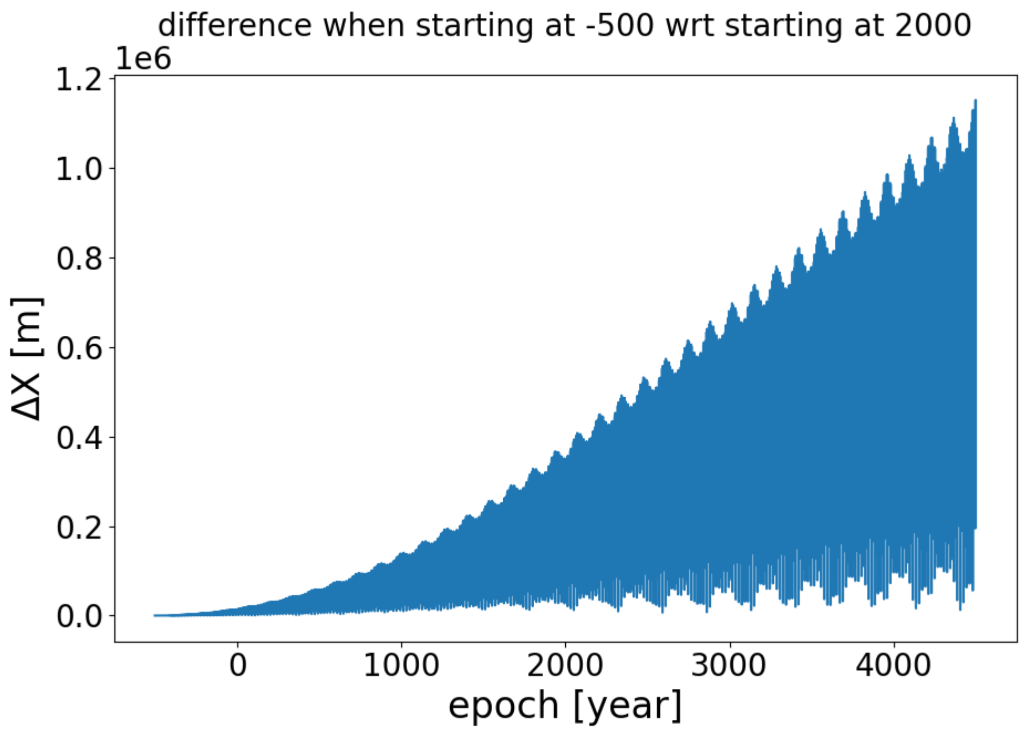


Figure 4.3: 5000-year run result, used to determine the effect of starting the run in the middle

the difference is still significantly smaller than that observed in the sensitivity analysis, by a factor of approximately 1000. This suggests that the starting point does not have the most substantial contribution to the overall error in the model.

Subsequently, the difference between starting with the JPL Horizons state at the starting point, or starting with the SPICE initial state was investigated. Since SPICE uses a slightly different initial state compared to JPL, the effect of this difference was examined to determine if it could be a dominant factor. The results are presented in Figure 4.4. Further discussion of these findings will be provided after the final figure is introduced.

Finally Figure 4.5 shows the difference between the JPL Horizons database and the model. To ensure a fair comparison, the model's initial state was adjusted to match the JPL state, changing the model's initial state from SPICE to JPL. The magnitude of the changing distance between the Earth and the Sun was calculated using two methods: the norm and the magnitude difference. The equations for these calculations are presented in Equation 4.1 for the norm and Equation 4.2 for the magnitude.

$$\Delta r_{norm} = ||\mathbf{r}_{Earth} - \mathbf{r}_{Sun}|| \quad (4.1) \qquad \Delta r_{magn} = ||\mathbf{r}_{Earth}|| - ||\mathbf{r}_{Sun}|| \quad (4.2)$$

The parameters used in these calculations include the difference calculated as a norm (Δr_{norm}), the distance calculated as a magnitude (Δr_{magn}), the distance between Earth and the Solar System Barycenter ($\mathbf{r}_{Earth}$), and the distance between the Sun and the Solar System Barycenter ($\mathbf{r}_{Sun}$). The norm provides a better understanding of the absolute distance between the Earth and the Sun, while the magnitude is more useful for comparison in validation, as it offers a clearer representation of the changing distance between objects. The norm, on the other hand, is more appropriate for assessing positional error.

In Figure 4.4, it can be seen that the order of difference - whether measured by norm, magnitude, or individual coordinates - is several orders of magnitude smaller than the difference between JPL and the model with the same initial state, as shown in Figure 4.5. This suggests that the initial state is unlikely to be the primary contributor to the discrepancy between the JPL database and the model. However,

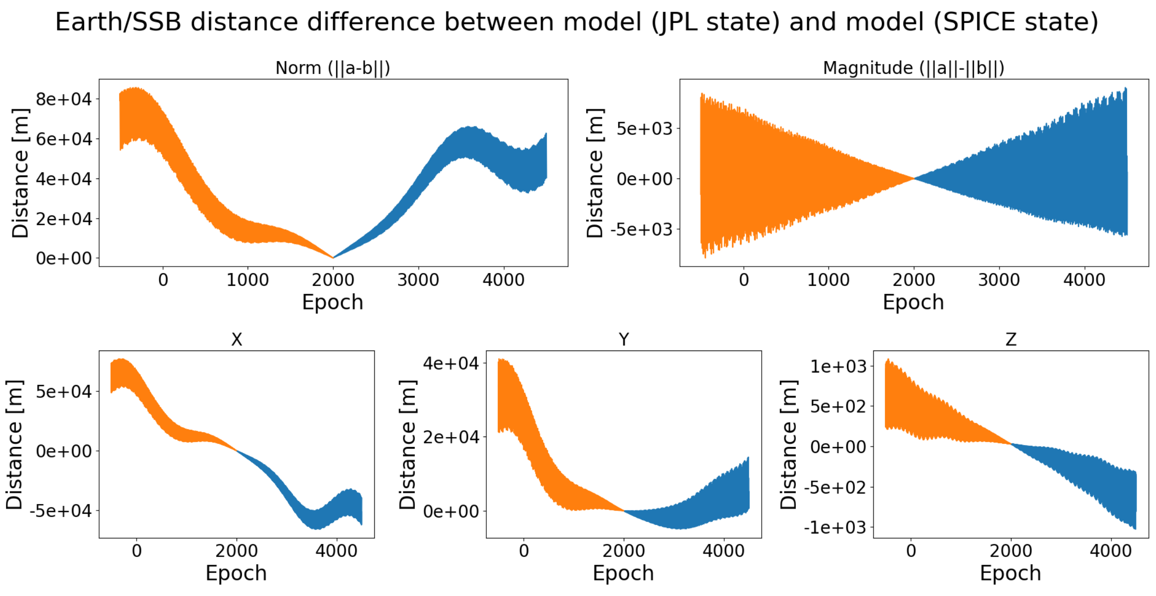


Figure 4.4: Initial state difference between SPICE and JPL. Orange shows the backward run, and blue represents the forward run. It is split in the norm (positional error), the magnitude (magnitude error), and in each coordinate.

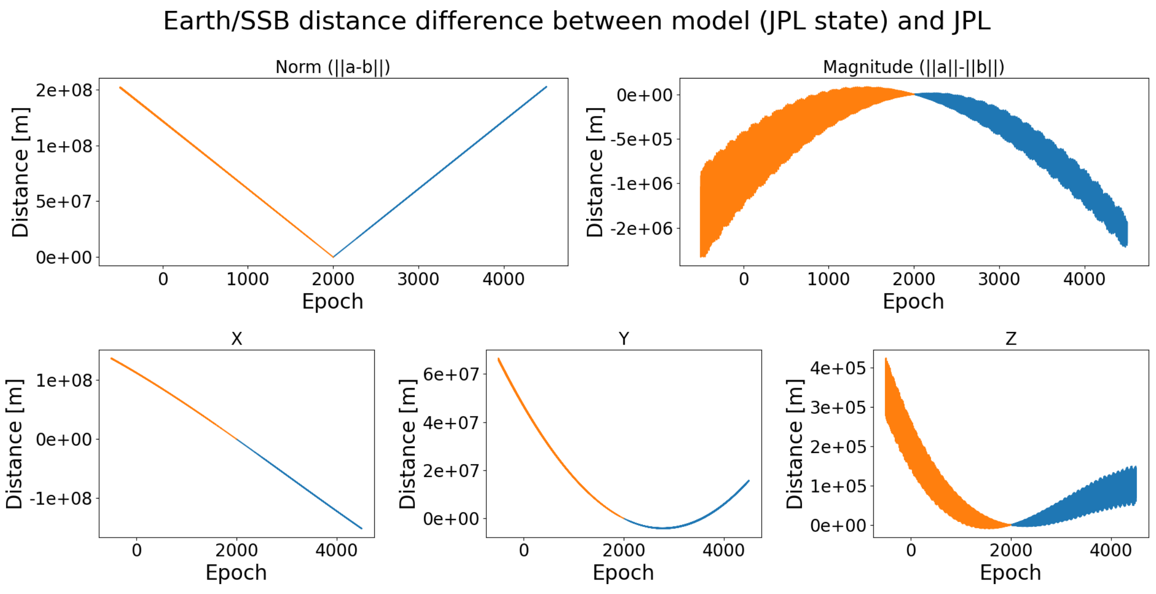


Figure 4.5: JPL and model difference with equal initial states. Orange shows the backward run, and blue represents the forward run. It is split in the norm (positional error), the magnitude (magnitude error), and in each coordinate.

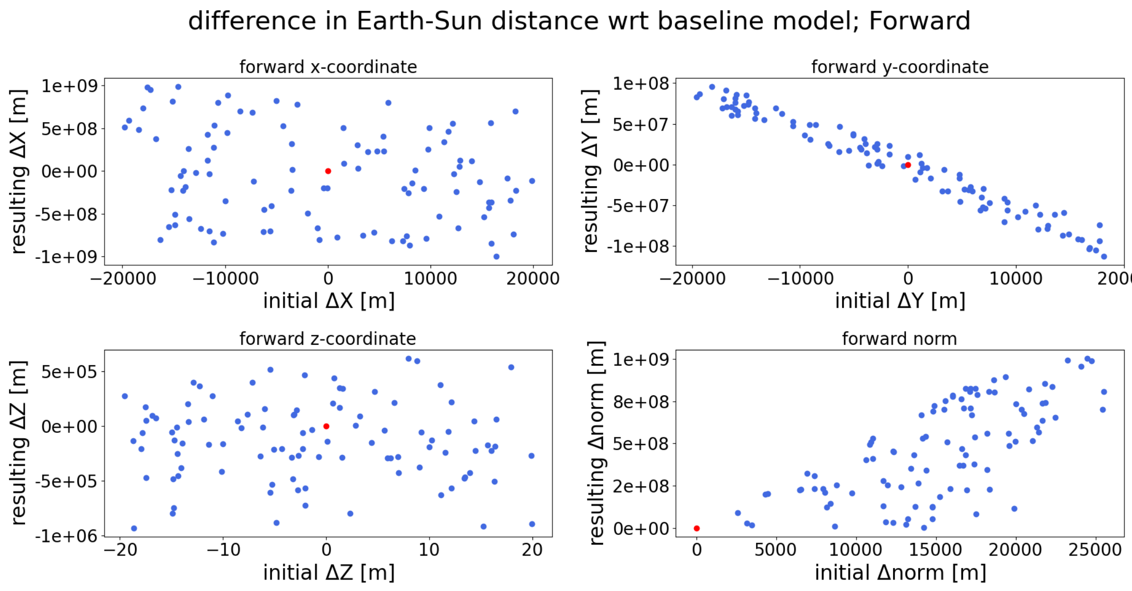


Figure 4.6: Sensitivity analysis of the forward runs. The red dot represents the baseline model without changes, and the 100 blue dots represent the 100 Monte-Carlo runs performed to determine the sensitivity. Shown are the X, Y, and Z-coordinates, and the norm of the differences.

the exact cause of the difference remains undetermined, as the methodology behind the JPL Horizons database was not found.

4.2. Sensitivity

The sensitivity analysis is a crucial aspect of model evaluation, as it helps assess the reliability of the results by examining how small variations in the initial conditions affect the overall outcome. The Monte Carlo method is used for the sensitivity analysis. In this approach, the initial state is randomly varied within defined limits, with separate independent adjustments made for the X, Y, and Z coordinates. This process is repeated multiple times, allowing for the creation of a scatterplot to identify any potential correlations. For this model, the sensitivity analysis was conducted 100 times, and the results can be found in Figure 4.6 for the forward run and Figure 4.7 for the backward run.

In both Figure 4.6 and Figure 4.7, it is observed that the X and Z coordinates show no correlation with the results. However, the Y coordinate exhibits a strong correlation, likely due to the fact that the Y coordinate reaches a maximum at the beginning of the analysis in terms of distance, as seen in Figure 5.1. Another important factor to address is the magnitude of the resulting difference, which reaches a maximum of around 1E9 meters - approximately 1% of the total distance between the Earth and the Sun. This indicates that the model is highly sensitive to changes in the initial state.

Given the uncertainty associated with this sensitivity, it is unlikely that a precise temperature value can be calculated, as the uncertainty would be too high to yield a reliable figure. However, it is essential to note that, despite this uncertainty, the overall trend observed in the model's results remains valid. Whatever initial state was used, the trend observed remained the same. Therefore, the conclusions drawn in Chapter 5 still hold, as confirmed by the sensitivity analysis.

More uncertainties could also be run through a sensitivity analysis, such as the masses of the astronomical bodies. This is, however, much more closely measured and observed, so the sensitivities would also be small. This analysis was however not performed, due to an expected small deviation from the result with the existing known margins and the time associated with preforming this analysis.

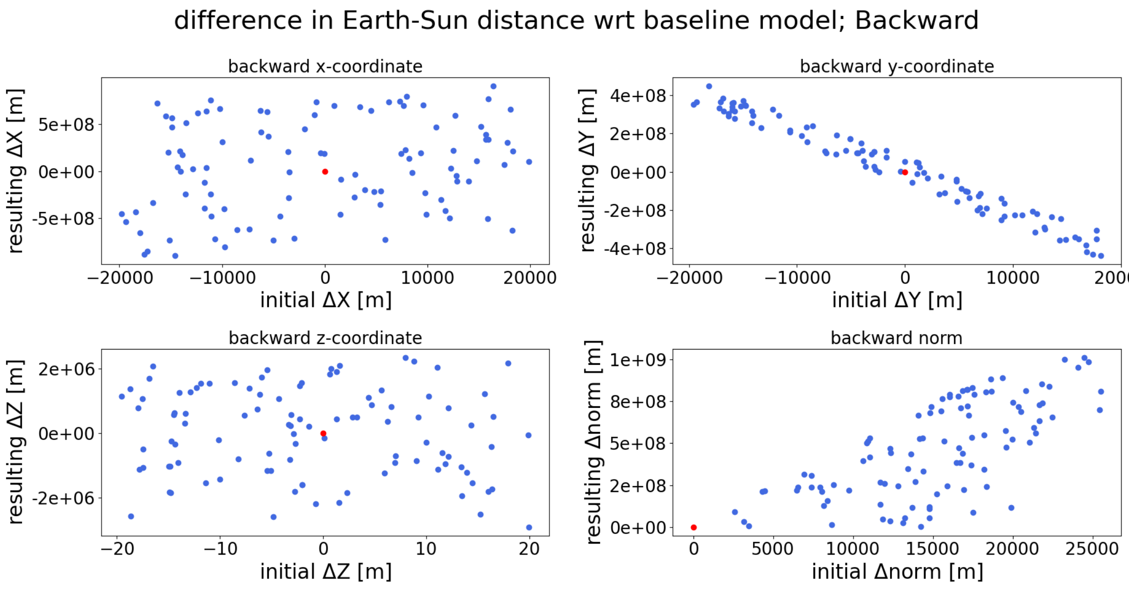


Figure 4.7: Sensitivity analysis of the forward runs. The red dot represents the baseline model without changes, and the 100 blue dots represent the 100 Monte-Carlo runs performed to determine the sensitivity. Shown are the X, Y, and Z-coordinates, and the norm of the differences.

5 Results

In this chapter, the results of the analysis performed by the algorithm and model are presented and discussed. First, Section 5.1 will focus on the results from the distance model, followed by an exploration of the hemispheric model results in Section 5.2. The findings from both models will then be combined and analyzed in Section 5.3. Finally, a direct comparison will be made with the result from V.V. Zharkova [43] in Section 5.4, and a conclusion will be drawn based on the results in Section 5.5.

5.1. Distance Model Results

In Figure 5.1, the result from the distance model are shown. Here, the orange line is the backward run (past), and the blue line is the forward run (future). The top two figures display the norm and the magnitude of the difference, while the bottom three show the results in X, Y, and Z coordinates.

First, the top left figure will be discussed, which shows the norm of the distance. It is apparent that there is a minimum present in the graph, corresponding to around the year 900. This minimum distance would translate to a maximum in temperature, instead of a decrease in temperature, as would be expected from V.V. Zharkova's research paper. A similar minimum is also observed in the top-right graph, which shows the magnitude of the difference. However, this minimum does not appear in any individual coordinate, indicating that it is not associated with any particular axis.

The relative thickness of the line in the norm-plot can be explained by the Solar Inertial Motion, where the sun rotates around the barycenter, as discussed in Section 2.2.1. The data in this plot is presented with a 1 Julian year interval, recorded on January 1st, and therefore, the result could also show the Milankovitch cycle relating to the eccentricity, where the perihelion and aphelion shift over time around the sun. However, this effect is expected to be minimal, as explained in Section 2.2.2. The effect should be particularly small at this point in the orbit, as it is close to perihelion.

However, it is also possible that the perihelion occurred around the year 900, which could explain the observed minimum in the graph. This graph thus does not completely disprove the statement from the V.V. Zharkova paper, stating that the distance between the Earth and the Sun is decreasing. However, subsequent results will provide further evidence against the paper's hypothesis.

5.2. Hemisphere Model Results

The results for the Hemisphere model discussed in Section 3.6 are presented in three figures: Figure 5.2 (by month), Figure 5.3 (by season), and Figure 5.4 (by year). The values for season and year are created by obtaining the value for each month and then averaging these out for the season and the year in total. These values therefore do not have the same pitfall as Figure 5.1, where only the data for the 1st of January was shown. It only shows the change of sunlight for the Northern Hemisphere, as discussed in Section 3.6.

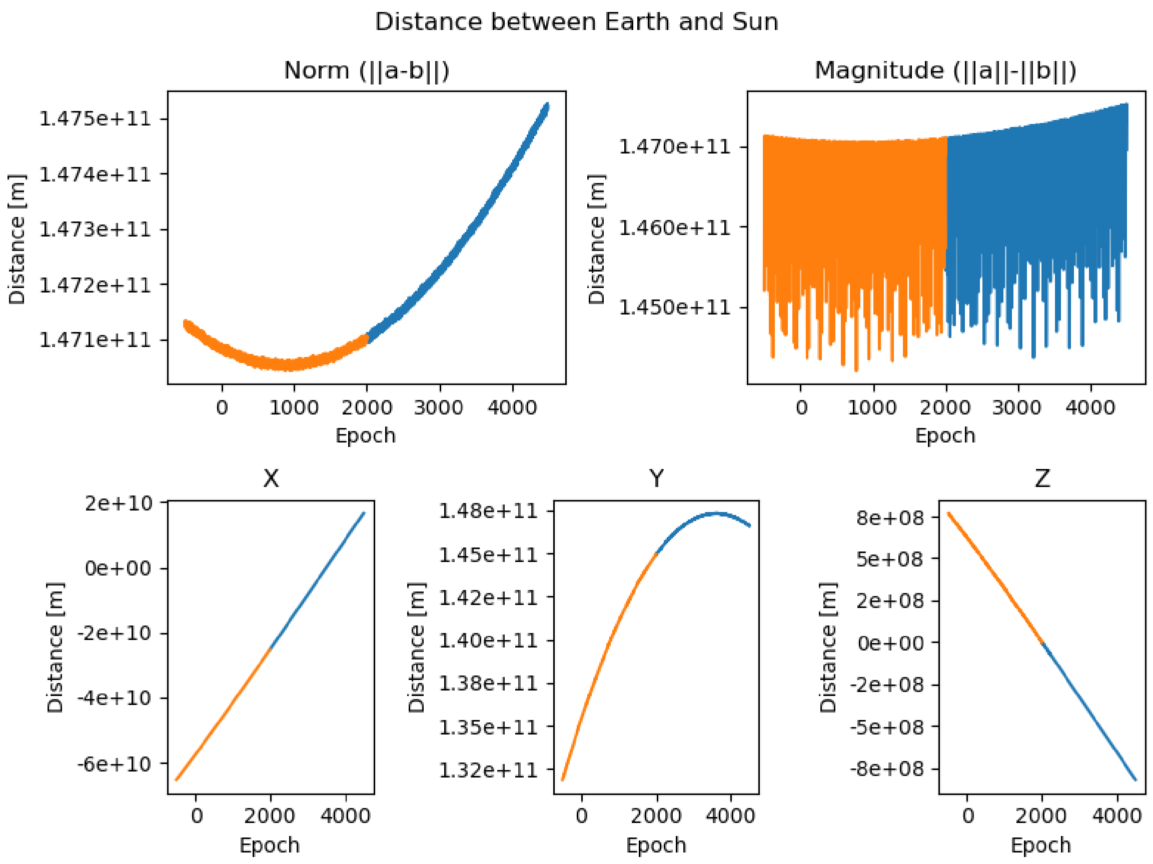


Figure 5.1: The result from the distance model. The orange line is the backward run (in the past), and the blue line is the forward run (to the future). The top two figures show the norm and the magnitude of the difference, while the bottom three show the results in X, Y, and Z-coordinates.

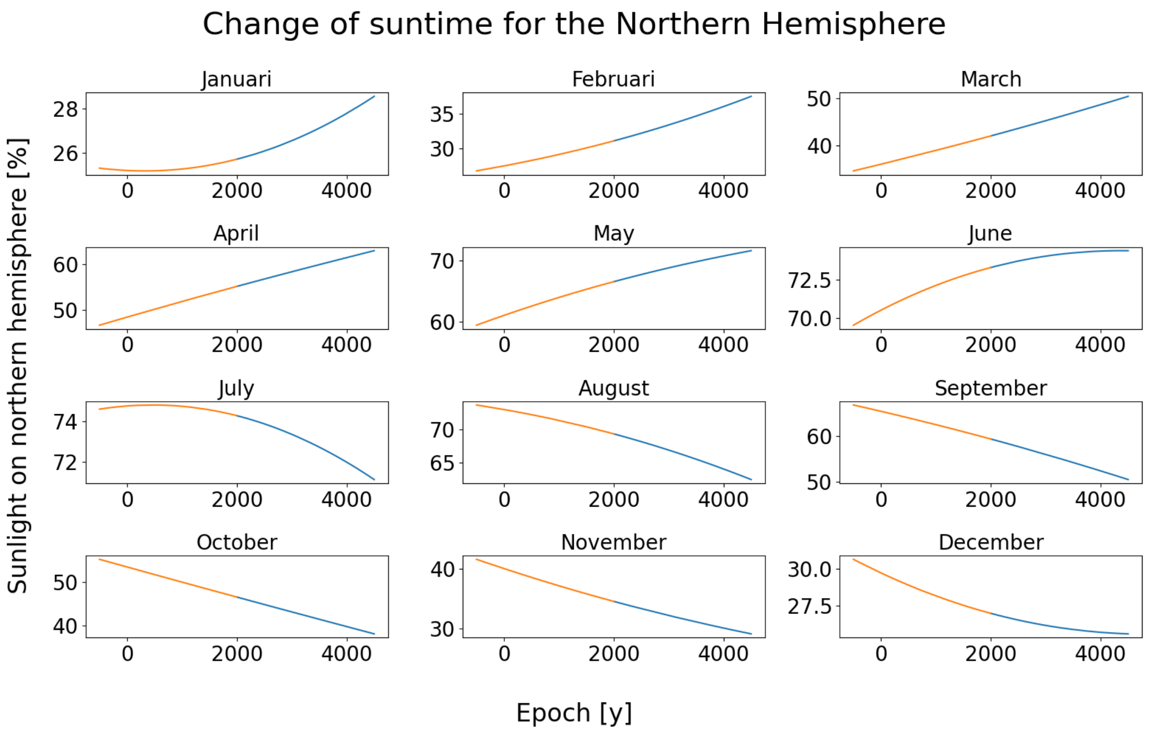


Figure 5.2: Change of sunlight for the Northern Hemisphere, shown per month. The blue line is the forward (future) run, and the orange line is the backward (past) run. Only the Northern Hemisphere data is shown, since the Southern Hemisphere is the opposite of that, and therefore does not add new insight. The seasons are defined by three moths: January, February, March for winter, April, May, June for spring, etc.

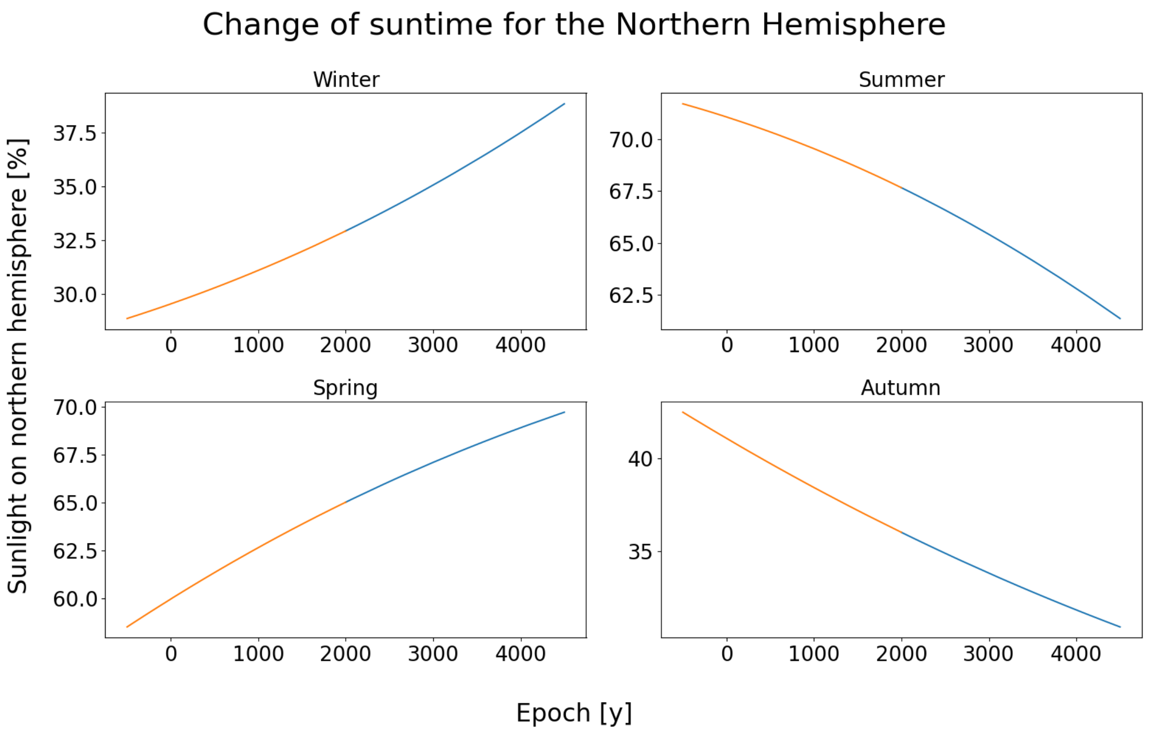


Figure 5.3: Change of sunlight for the Northern Hemisphere, shown per season. The blue line is the forward (future) run, and the orange line is the backward (past) run. Only the Northern Hemisphere data is shown, since the Southern Hemisphere is the opposite of that, and therefore does not add value.

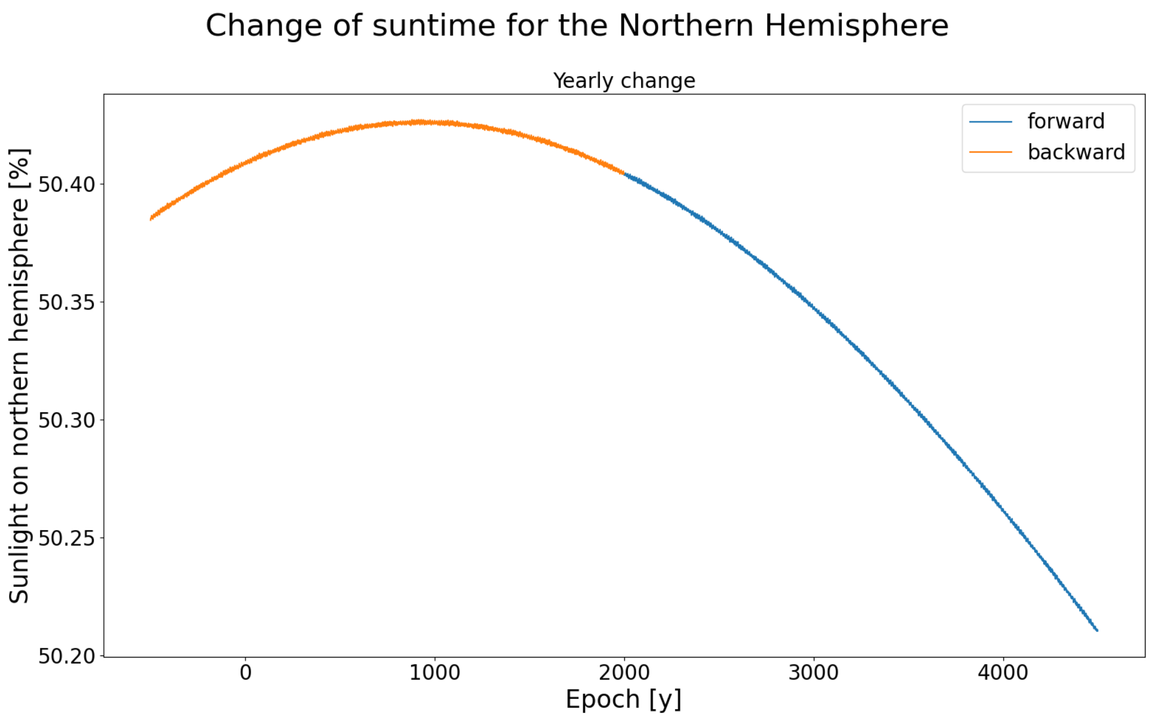


Figure 5.4: Change of sunlight for the Northern Hemisphere, shown per year. The blue line is the forward (future) run, and the orange line is the backward (past) run. Only the Northern Hemisphere data is shown, since the Southern Hemisphere is the opposite of that, and therefore does not add value.

First, Figure 5.2 will be discussed. It shows that sunlight levels in the Northern Hemisphere change over the course of the year. From January to June, the percentage of sunlight increases, while from July to December, it decreases in the Northern Hemisphere. This pattern only partially supports V.V. Zharkova's claim that sunlight increases during the summer months, resulting in more energy being absorbed by Earth.

Next, in Figure 5.3, a similar trend is observed: In winter and spring, the percentage of sunlight in the Northern Hemisphere is increasing, while for summer and autumn, the percentage of sunlight in the Northern Hemisphere is decreasing. However, this contradicts the statements of V.V. Zharkova, which states that the Northern Hemisphere experiences an increase in sunlight during its warmer months. Instead, we observe the opposite: the Northern Hemisphere experiences a decrease in sunlight during its warmer season. Similarly, the Southern Hemisphere sees a decrease in the percentage of sunlight in its warmer period, which corresponds to winter in the Northern Hemisphere. While this is not directly shown in the graph, it can be inferred since when the percentage of sunlight increases in the Northern Hemisphere, it decreases in the Southern Hemisphere, and vice versa.

Finally, in Figure 5.4 which presents the yearly averages, the same conclusion is drawn from the seasonal data: the percentage of sunlight in the Northern Hemisphere is decreasing, rather than increasing. This decrease in sunlight results in higher reflectivity of the Earth, which would have a cooling effect on global temperatures. Therefore, according to these patterns, the temperature should be decreasing, which contradicts the observed increase in temperature today, so these are not related.

5.3. Combined Model Results

In this section, the combined model result will be discussed. The combination process involves using the distance calculated by the distance model to determine the solar intensity on Earth, applying the inverse square law (Equation 2.1). This allows for the calculation of sunlight that falls on each Hemisphere, producing an average sunlight intensity per Hemisphere, including areas experiencing

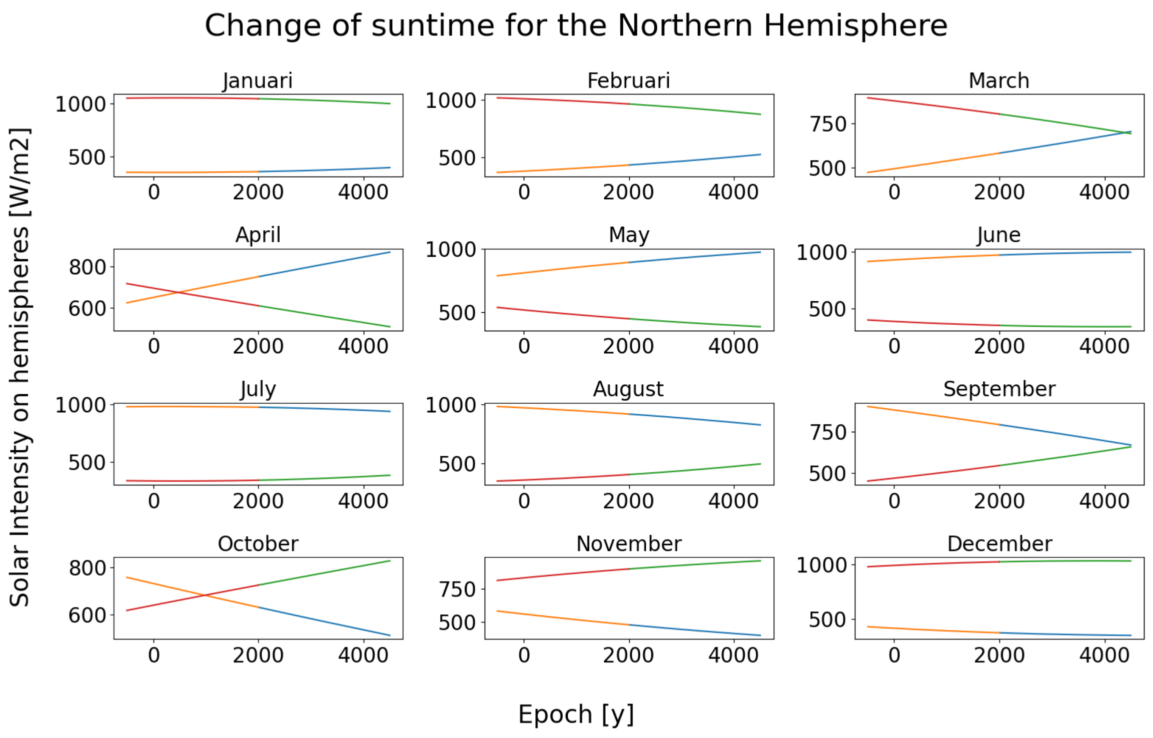


Figure 5.5: Change of solar intensity for both Hemispheres, shown per month. The orange/blue line is the Northern Hemisphere, while the green/red line is the Southern Hemisphere. It is split into the backward (past) run (orange for the Northern Hemisphere and red for the Southern Hemisphere) and the forward (future) run (blue for the Northern Hemisphere and green for the Southern Hemisphere).

nighttime. These results, like those in Section 5.2, are presented for each month, as season averages, and as yearly averages in Figure 5.5, Figure 5.6, and Figure 5.7, respectively.

First, the focus is on Figure 5.5, which shows the monthly changes in solar intensity. The orange and blue lines represent the Northern Hemisphere, while the red and green lines represent the Southern Hemisphere. Since solar intensity is factored into the model at this step, the two Hemispheres are not necessarily perfect opposites of each other. However, the general trends observed are similar to those in Figure 5.2. From January to June, the Northern Hemisphere experiences an increase in solar intensity, while from July to December, it experiences a decrease. In contrast, the Southern Hemisphere follows the opposite pattern: a decrease in sunlight intensity from January to June, and an increase from July to December.

Moving on to Figure 5.6, the same trend is observed again: for both Hemispheres, in their respective summer period, the solar intensity decreases. This contradicts the conclusions presented by V.V. Zharkova. Furthermore, when observing Figure 5.7, the yearly averages strengthen this observation. The solar intensity for both Hemispheres is slowly decreasing over the course of the year, which runs counter to Zharkova's findings.

Based on the results obtained from the algorithm, both the distance model and the hemispheric model suggest that Earth's temperature should be decreasing, rather than increasing, due to the observed changes in solar intensity and the Earth-Sun distance.

5.4. Zharkova Comparison Results

Finally, to better compare the model and algorithm used in this study with the model employed in V.V. Zharkova's research [43], a graph directly taken from that paper is reproduced, or the way they are presented. This allows us to determine whether the difference lies in the models themselves or merely in the results produced. The graph taken from V.V. Zharkova is Figure 5.8, which illustrates the distance between the Earth and the Sun over the first six months, spanning a period of six years from 600 to

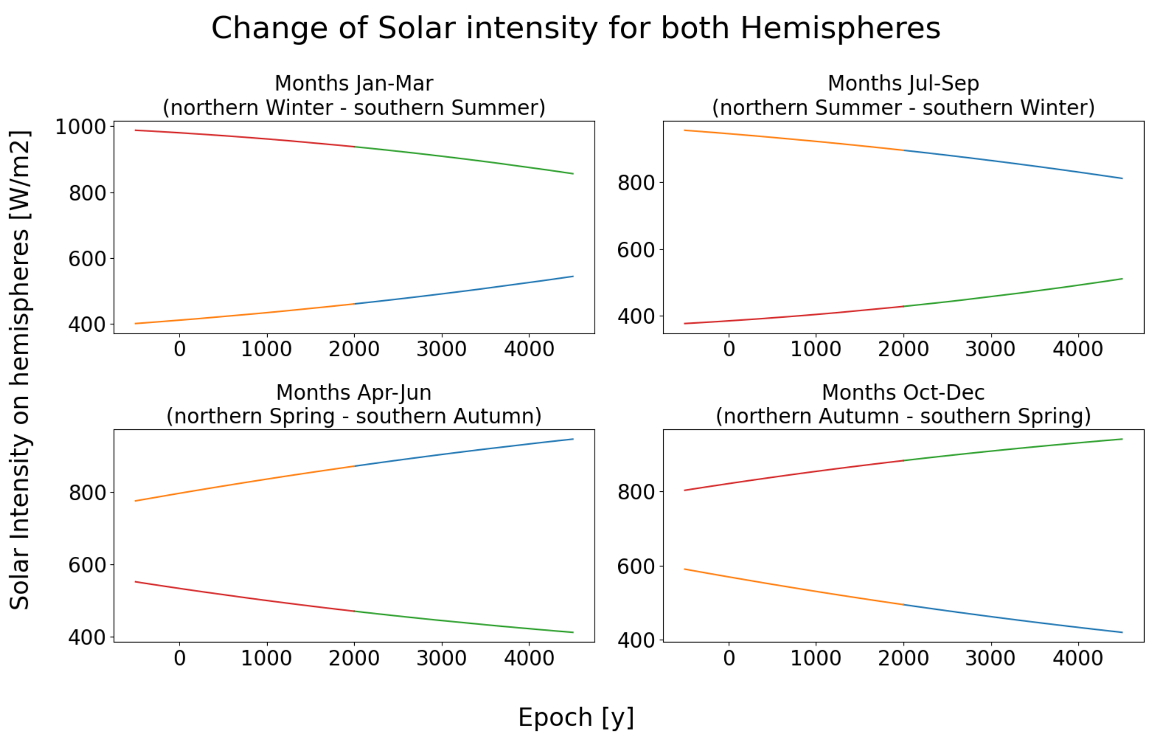


Figure 5.6: Change of solar intensity for both Hemispheres, shown per season. The orange/blue line is the Northern Hemisphere, while the green/red line is the Southern Hemisphere. It is split into the backward (past) run (orange for the Northern Hemisphere and red for the Southern Hemisphere) and the forward (future) run (blue for the Northern Hemisphere and green for the Southern Hemisphere).

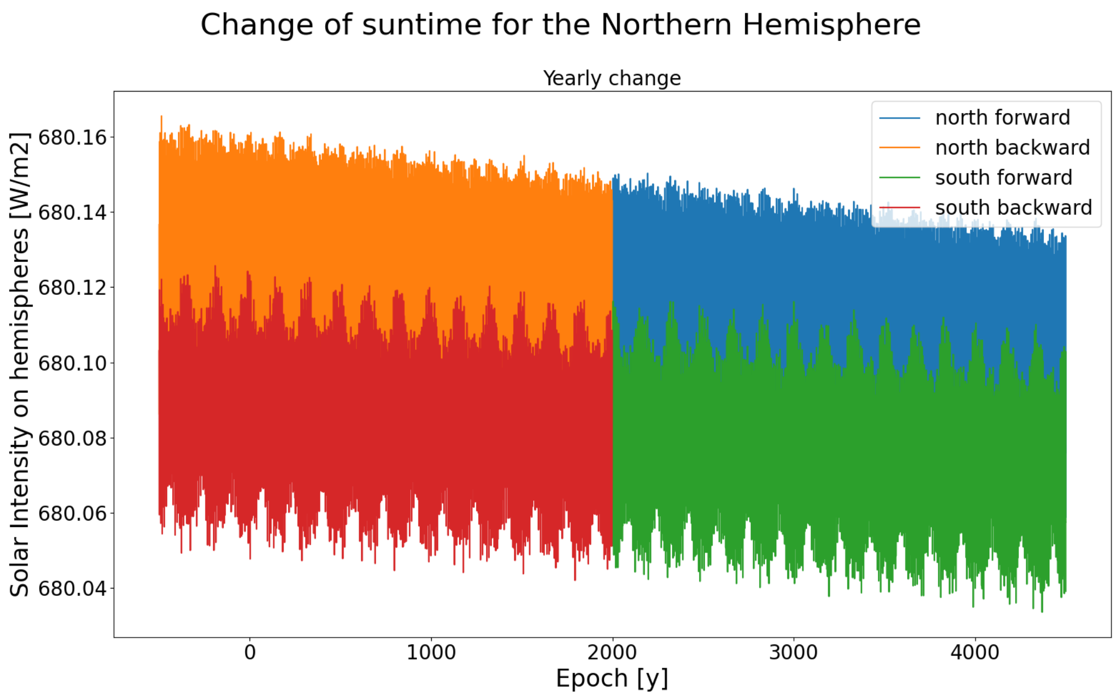


Figure 5.7: Change of solar intensity for both Hemispheres, shown per year. The orange/blue line is the Northern Hemisphere, while the green/red line is the Southern Hemisphere. It is split into the backward (past) run (orange for the Northern Hemisphere and red for the Southern Hemisphere) and the forward (future) run (blue for the Northern Hemisphere and green for the Southern Hemisphere).

2600. It is evident that the distance decreases over time during these months, as shown by the changes between the different lines, each representing a different year. This decrease in distance potentially signals an increase in solar intensity, leading to higher temperatures due to increased solar irradiance.

This graph is reproduced in Figure 5.9, using data from our model. It was expected to not follow Zharkova's graph closely, however, as can be observed, it is clear that they are very similar. This suggests that Zharkova's model closely matches the outcomes of the model used in this report during the time period depicted in the graphs. This similarity appears to contradict earlier results, but it can still be explained. The graph here focuses only on the first six months, during which the Earth-Sun distance decreases. However, when linked to the other results shown here, even if the distance decreases in these months, the total solar irradiance over the entire year still shows a decreasing trend.

The close similarity between the models only strengthens the conclusion that Zharkova's hypothesis is flawed, as the two models — despite being based on entirely independent methods (likely, since Zharkova did not share model specifics) — still yield consistent results. However, this similarity does weaken the claim regarding the minimum at the specified location in the model's graph. It is likely that this minimum would occur on different dates depending on the specific time period shown in such a distance graph.

5.5. Result Conclusion

In Section 5.1, the results from the distance model were discussed, revealing that the Earth-Sun distance increases rather than decreases. However, as later analysis showed, this local minimum distance is likely to shift depending on the specific date considered. In Section 5.2, the results from the Hemisphere model were examined. Both Hemispheres experience a decrease in sunlight during their respective summer months, which would be expected to lead to lower temperatures.

Further, when combining the results in Section 5.3, it was observed that the average solar intensity for both Hemispheres is actually decreasing, which contradicts the findings of V.V. Zharkova. Direct comparisons of graphs from the two models in Section 5.4 show a close correlation, leading to the conclusion that Zharkova's model closely matches the model used here. This suggests that the interpretation of results can be highly dependent on how the data is presented.

However, from Figure 5.7 leaves no room for doubt: according to the model used in this paper, solar intensity is decreasing.

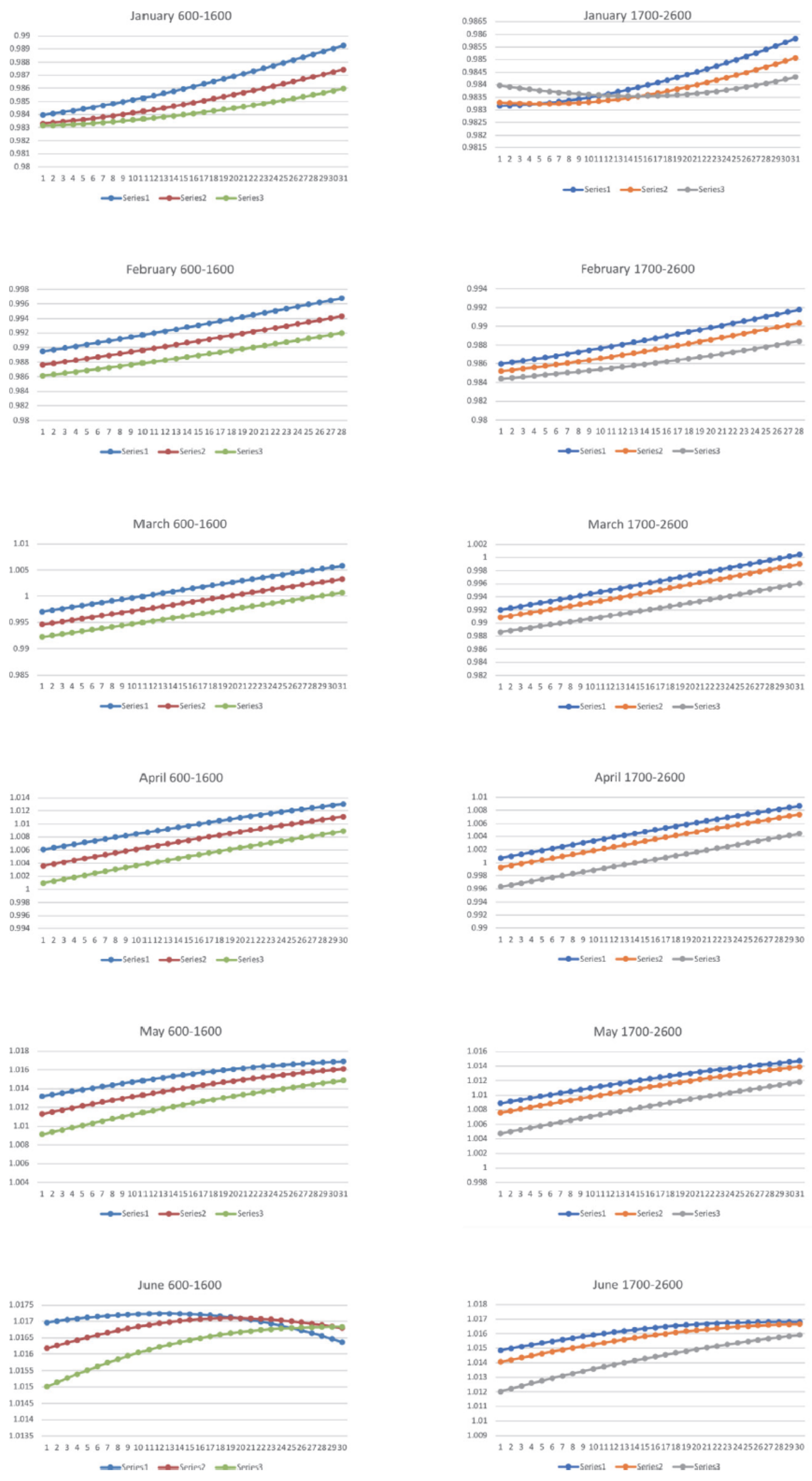


Figure 5.8: Figure from V.V. Zharkova [43], showing the changing distance between the Earth and the Sun. It has the caption: ”Variations of the sun-earth distances (in astronomical units, au) versus days of the month (X-axis) in January–June for three sample years in the millennium M1 (600–1600) (left) and M2 (1600–2600) (right). Left column: Blue - year 600, red - 1100 and green - 1600; right column: Blue - year 1700, red - 2020 and grey - 2600.”

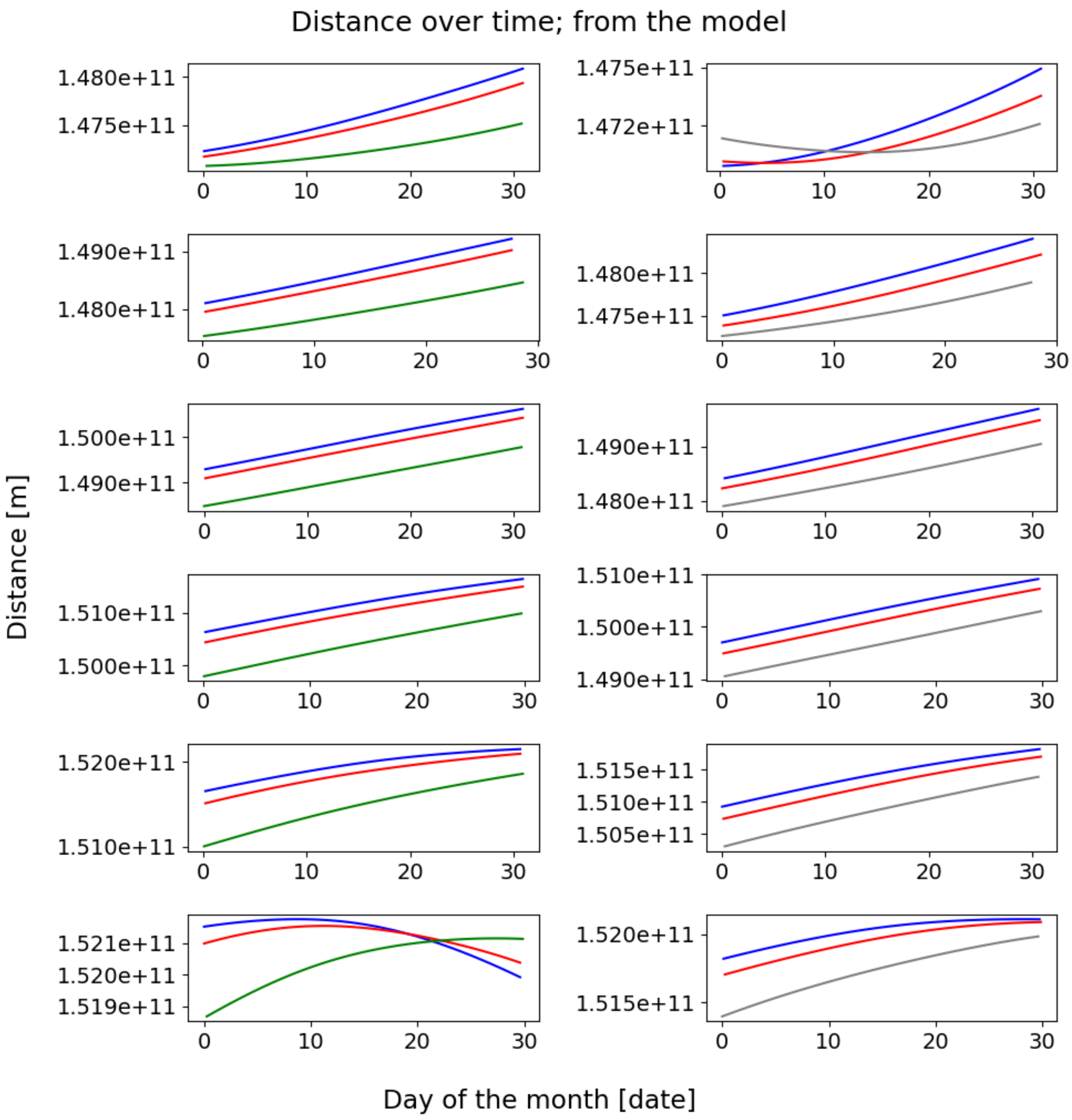


Figure 5.9: Recreated Figure 5.8 using model data, showing the changing distance between the Earth and the Sun. The years are in; the left column: blue - 600, red - 1100, and green - 1600; the right column: blue - 1700, red - 2020, and grey - 2600

6 Conclusion

In this chapter, a conclusion will be made in Section 6.1. After that, recommendations will be provided for both improvements to the results shown and further research additional to the analysis performed in this report.

6.1. Conclusion

At the beginning of this report, in Chapter 1, several research questions were introduced. After providing background information in Chapter 2 and describing the methodology in Chapter 3, a sensitivity analysis was performed, along with a detailed discussion of the verification and validation methods used in this report in Chapter 4. Following that, the results were presented and discussed in Chapter 5. Now, based on all the data presented, a conclusion can be drawn, and the research questions can be answered.

To answer the first subquestion - *"Can the results as seen in V.V. Zharkova [43] be reproduced without changing the parameters of the solar system, with only a simplified model of solar variation?"* - the model described in Chapter 3 was used. As shown in the comparison between Figure 5.8 and Figure 5.9, the results were successfully reproduced.

The second subquestion - *"What is the difference between the model used by this paper and the model used by V.V. Zharkova?"* - can be addressed with Figure 5.1. It is evident from the figure that the model used in this study predicts an increase in the distance between the Earth and the Sun, in contrast to the results observed from V.V. Zharkova's study.

For the final subquestion - *"Can the hemispheric effects (as described by V.V. Zharkova [43]) be quantified in a simplified Earth hemispheric model?"* - the answer is provided by Figure 5.4. While it is possible to quantify the hemispheric effects in a simplified model, the results contradict those reported by Zharkova. Rather than observing the predicted warming effects, the model shows a cooling effect in the Northern Hemisphere during the summer months, opposing the conclusions drawn by Zharkova and colleagues.

These answers to the subquestions lead to the main research question — *"How does the current perceived change in the distance between Earth and the Sun affect the global temperature?"* - which can be addressed by referring to the previous subquestion answers as well as Figure 5.7. The findings indicate that the Earth is experiencing a natural cooling effect due to a decrease in solar intensity, rather than the warming predicted by V.V. Zharkova's research and observed in current trends. However, the exact difference with V.V. Zharkova's paper [43] can not be precisely found, since Zhakova also includes other factors, such as the magnetic field activity of the Sun. The statements that either the Earth-Sun distance or Earth's Hemispheres are (partly) responsible have however been rejected.

In conclusion, the results of this study do not support Zharkova's conclusions. Based on the evidence presented, it is clear that the model used in this paper predicts a cooling trend, rather than the warming trend suggested by V.V. Zharkova [43].

This cooling trend means that the human factor in the warming of the Earth observed today is even greater than currently understood since the cooling effect also needs to be negated by human influences. The magnitude of the negated cooling effect and thus the increased influence of humankind depends on the actual cooling effect, which unfortunately could not be established by this report.

6.2. Recommendations

The most critical recommendation is to address the uncertainty highlighted throughout the analysis. Due to the high uncertainty observed in both the sensitivity analysis and the comparison with the JPL Horizons database, it is not possible to reliably calculate a specific temperature value. This uncertainty limits the conclusions that can be drawn from the model's results. By improving the model's accuracy, it would be possible to calculate a more precise estimate of the cooling effects identified in this study. Such an improvement would enhance our understanding of the potential impact of natural solar variations on climate and enable more accurate assessments of human-induced climate change.

After reducing uncertainty, a further important step would be to estimate the albedo value for each hemisphere and calculate its effect on temperature. A more precise albedo estimate would provide additional insight into the factors influencing Earth's climate and could help clarify the role of human activities in altering climate patterns.

Another area for improvement lies in the selection of integrators and propagators. Although the selection process was conducted adequately, there is room for refinement, particularly with the choice of propagator. One potential improvement would be to allow multiple integrators to perform the analysis over the entire timespan of the model, rather than just for a limited timeframe. By examining the maximum error across the full period, this approach could confirm the correctness of the integrator selection process and potentially reduce errors, while also improving CPU efficiency.

Regarding the spherical harmonics used in the model, the current approach applies them universally to all bodies in the solar system. However, it may be more efficient to use a selective approach—applying spherical harmonics only to the bodies that are most affected by them. For instance, instead of applying Jupiter's spherical harmonics to all objects in the solar system, it could be limited to the Galilean moons, which are much more directly influenced by Jupiter's gravity. This selective application of spherical harmonics could significantly optimize performance. By only including the harmonics for bodies that are substantially affected by their gravitational fields, the model could avoid unnecessary computational overhead for distant bodies like Neptune or Titan. This approach would likely improve model accuracy, as it would focus computational resources where they are most needed, while still maintaining the overall integrity of the results without introducing excessive computational cost. This selective use could improve the model's accuracy without significantly increasing CPU time, as seen in the current implementation.

Further research into the discrepancy between the model used in this paper and the JPL Horizons database is also highly recommended. At the time of writing, the specific methodology behind the JPL Horizons database was not found. Understanding the sources of the observed differences—particularly why the discrepancy is so significant—could reduce the error margin and lead to more precise model results. More importantly, such research would increase confidence in the model's accuracy, which would, in turn, strengthen the conclusions drawn in this paper. Investigating the methodology behind the JPL Horizons database would provide a deeper understanding of its data sources, assumptions, and potential limitations. By comparing these with the assumptions made in this model, it would be possible to identify the root causes of discrepancies and determine whether they are due to differences in model assumptions, data accuracy, or other factors.

In addition, further exploration of other factors suggested by V.V. Zharkova, such as solar activity and the solar magnetic field, would be beneficial. These effects, which were excluded for simplicity in the current model, could potentially have a significant impact on the model's results. Solar activity, such as sunspots and solar flares, affects the amount of solar radiation reaching the Earth. Similarly, the solar magnetic field can influence space weather, affecting the Earth's atmosphere and possibly altering the energy balance. By including these factors, the model would be able to better account for short- and long-term variations in solar energy and their potential impact on Earth's climate.

Finally, additional research into the presentation of results, particularly concerning the Earth-Sun distance, could help clarify how the way data is displayed influences the model's conclusions. While this may be a less pressing concern compared to the other recommendations, it could still provide valuable insights into the presentation of scientific data. For example, comparing different visual representations of the Earth-Sun distance could help determine which format best communicates the underlying trends in the data, and whether certain presentations might inadvertently obscure or misrepresent important details.

References

[1] NASA’s Jet Propulsion Laboratory Alan Buis. *Milankovitch (Orbital) Cycles and Their Role in Earth’s Climate*. 2020. URL: `https://climate.nasa.gov/news/2948/milankovitch-orbital-cycles-and-their-role-in-earths-climate/` (visited on 02/20/2024).

[2] A. Berger. “Milankovitch Theory and climate”. In: *Reviews of Geophysics* 26.4 (1988), pp. 624–657. DOI: `https://doi.org/10.1029/RG026i004p00624`.

[3] Gerald Maurice Clemence. “The Relativity Effect in Planetary Motions”. In: *Reviews of Modern Physics* 19 (1947), pp. 361–364.

[4] Matija Ćuk et al. “Long-Term Evolution of the Saturnian System”. In: *Space Science Reviews* 220.2 (2024). DOI: `10.1007/s11214-024-01049-2`.

[5] NASA; by D.R. Williams. *Earth Fact Sheet*. 2021. URL: `https://nssdc.gsfc.nasa.gov/planetary/factsheet/earthfact.html` (visited on 08/24/2022).

[6] NASA; by D.R. Williams. *Jupiter Fact Sheet*. 2024. URL: `https://nssdc.gsfc.nasa.gov/planetary/factsheet/jupiterfact.html` (visited on 02/24/2024).

[7] NASA; by D.R. Williams. *Mars Fact Sheet*. 2024. URL: `https://nssdc.gsfc.nasa.gov/planetary/factsheet/marsfact.html` (visited on 02/25/2024).

[8] NASA; by D.R. Williams. *Mercury Fact Sheet*. 2024. URL: `https://nssdc.gsfc.nasa.gov/planetary/factsheet/mercuryfact.html` (visited on 02/22/2024).

[9] NASA; by D.R. Williams. *Neptune Fact Sheet*. 2024. URL: `https://nssdc.gsfc.nasa.gov/planetary/factsheet/neptunefact.html` (visited on 02/29/2024).

[10] NASA; by D.R. Williams. *Saturn Fact Sheet*. 2024. URL: `https://nssdc.gsfc.nasa.gov/planetary/factsheet/saturnfact.html` (visited on 02/26/2024).

[11] NASA; by D.R. Williams. *Venus Fact Sheet*. 2024. URL: `https://nssdc.gsfc.nasa.gov/planetary/factsheet/venusfact.html` (visited on 02/22/2024).

[12] Rola Dbouk and Jack Wisdom. “The Origin of Jupiter’s Obliquity”. In: *Planetary Science Journal* 4.10 (2023). DOI: `10.3847/PSJ/acf9f8`.

[13] A. Einstein. “The Foundation of the General Theory of Relativity”. In: *Annalen der Physik* (1916). translated by: Satyendra Nath Bose.

[14] W.M. Folkner. *Uncertainties in the JPL planetary ephemeris*. 2010.

[15] Grant Foster et al. “Comment on “Heat capacity, time constant, and sensitivity of Earth’s climate system” by S. E. Schwartz”. In: *Journal of Geophysical Research: Atmospheres* 113.D15 (2008). DOI: `https://doi.org/10.1029/2007JD009373`.

[16] P. R. Goode et al. “Earth’s Albedo 1998–2017 as Measured From Earthshine”. In: *Geophysical Research Letters* 48.17 (2021), e2021GL094888. DOI: `https://doi.org/10.1029/2021GL094888`.

[17] P. R. Goode et al. “Earthshine observations of the Earth’s reflectance”. In: *Geophysical Research Letters* 28.9 (2001), pp. 1671–1674. DOI: `https://doi.org/10.1029/2000GL012580`.

[18] John R. Gribbin. *Deep Simplicity Chaos, Complexity and the Emergence of Life*. 2004.

[19] James Hansen et al. “Earth’s Energy Imbalance: Confirmation and Implications”. In: *Science* 308.5727 (2005), pp. 1431–1435. DOI: `10.1126/science.1110252`.

[20] Stuart A. Harris. “Comparison of Recently Proposed Causes of Climate Change”. In: *Atmosphere* 14.8 (2023). ISSN: 2073-4433. DOI: `10.3390/atmos14081244`.

[21] J. Horner and B.W. Jones. “Jupiter – friend or foe? I: The asteroids”. In: *International Journal of Astrobiology* 7.3–4 (July 2008), pp. 251–261. ISSN: 1475-3006. DOI: `10.1017/s1473550408004187`.

[22] X. Y. Hou, D. J. Scheeres, and L. Liu. “Saturn Trojans: a dynamical point of view”. In: *Monthly Notices of the Royal Astronomical Society* 437.2 (Nov. 2013), pp. 1420–1433. ISSN: 0035-8711. DOI: 10.1093/mnras/stt1974.

[23] Michael R. House. “Orbital forcing timescales: an introduction”. In: *Geological Society, London, Special Publications* 85.1 (1995), pp. 1–18. DOI: 10.1144/GSL.SP.1995.085.01.01.

[24] L. Iess et al. “Measurement of Jupiter’s asymmetric gravity field”. In: *Nature* 555 (2018), pp. 220–222. DOI: 10.1038/nature25776.

[25] Lorenzo Iorio. “Constraints on the location of a putative distant massive body in the Solar System and on the External Field Effect of MOND from recent planetary data”. In: *Celestial Mechanics and Dynamical Astronomy* 112 (Feb. 2011). DOI: 10.1007/s10569-011-9386-7.

[26] IPCC et al. “Climate change 2023: Synthesis report.” In: *IPCC, Geneva, Switzerland* (July 2023). DOI: 10.59327/ipcc/ar6-9789291691647.

[27] J. Laskar and M. Gastineau. “Existence of collisional trajectories of Mercury, Mars and Venus with the Earth”. In: *Nature* 459.7248 (June 2009), pp. 817–819. DOI: 10.1038/nature08096.

[28] J. Laskar et al. “La2010: a new orbital solution for the long-term motion of the Earth”. In: *Astronomy & Astrophysicss* 532 (July 2011), A89. DOI: 10.1051/0004-6361/201116836.

[29] J. Laskar et al. “Long term evolution and chaotic diffusion of the insolation quantities of Mars”. In: *Icarus* 170.2 (2004), pp. 343–364. ISSN: 0019-1035. DOI: https://doi.org/10.1016/j.icarus.2004.04.005.

[30] Jacques Laskar. “A numerical experiment on the chaotic behaviour of the Solar System”. In: *Nature* 338 (1989), pp. 237–238.

[31] Jack Jonathan Lissauer and Imke De Pater. *Fundamental Planetary Sciences: Physics, Chemistry and Habitability*. Cambridge University Press, 2019.

[32] Zinovy Malkin and Natalia Miller. “Chandler wobble: Two more large phase jumps revealed”. In: *Earth Planets and Space* 62 (Aug. 2009). DOI: 10.5047/eps.2010.11.002.

[33] T. Markvart and Luis Castañer. *Practical Handbook of Photovoltaics: Fundamentals and Applications*. Elsevier Advanced Technology, 2003.

[34] Gerald E. Marsh. *Irradiance Variations due to Orbital and Solar Inertial Motion: The Effect on Earth’s Surface Temperature*. 2020. arXiv: 2003.01374 [astro-ph.EP].

[35] NASA. *What Is the Solar Cycle?* 2021. URL: https://spaceplace.nasa.gov/solar-cycles/en/#:~:text=The%20Short%20Answer%3A,generating%20a%20powerful%20magnetic%20field. (visited on 02/19/2024).

[36] Andrew C. Overholt, Adrian L. Melott, and Martin Pohl. “Testing the link between terrestrial climate change and galactic spiral arm transit”. In: *The Astrophysical Journal* 705.2 (Oct. 2009), pp. L101–L103. ISSN: 1538-4357. DOI: 10.1088/0004-637x/705/2/l101.

[37] Nicola Scafetta. “Solar Oscillations and the Orbital Invariant Inequalities of the Solar System”. In: *Solar Physics* 295 (2020). URL: https://api.semanticscholar.org/CorpusID:255065994.

[38] Ernst Schrama. *Dynamics part 1*. Lecture material, Satellite Orbit Determination (AE4872). 2023.

[39] Scott S. Sheppard and Chadwick A. Trujillo. “A survey for satellites of Venus”. In: *Icarus* 202.1 (July 2009), pp. 12–16. ISSN: 0019-1035. DOI: 10.1016/j.icarus.2009.02.008.

[40] NOAA; by Space Weather Prediction Center. *Solar cycle progression*. 2024. URL: https://www.swpc.noaa.gov/products/solar-cycle-progression#:~:text=The%20Prediction%20Panel%20predicted%20Cycle,November%202024%20and%20March%202026.&text=SWPC%20Space%20Weather%20Operations%20(SWO)%2C%20Daily%20Observations. (visited on 02/19/2024).

[41] TUDAT development Team. *Default environment models - Gravity field*. 2024. URL: https://docs.tudat.space/en/latest/_src_user_guide/state_propagation/environment_setup/default_env_models.html (visited on 11/04/2024).

[42] David Tytell. *Why doesn’t Venus have a moon?* 2006. URL: https://skyandtelescope.org/astronomy-news/why-doesnt-venus-have-a-moon/ (visited on 02/22/2024).

[43] V.V. Zharkova et al. “Oscillations of the baseline of solar magnetic field and solar irradiance on a millennial timescale”. In: *Scientific Reports* 9.1 (2019). *note: this article has been redacted, but is still contested by most authors and defended in further works. Therefore it is deemed relevant and included in this report*.

A
code files

main.py

```python
# Load files
from settings import *
import utilities
import model as model
import verification
import validation
import plot

# Load modules
import time

time_start = time.time()
time_struct = time.localtime()
print(f"run␣starting␣at␣{time_struct[3]}:{time_struct[4]}:{time_struct[5]}\n")

# get JPL Earth data
JPL_state_Earth = utilities.get_JPL_Earth_Data()

def long_run(backward=False):
    system_state_array_forward = model.subsequent_runs()

    model_error_dict_forward = validation.get_accuracy(system_state_array_forward,
        JPL_state_Earth, types=['last', 'max'])
    for i in model_error_dict_forward:
        print(f"distance␣forward␣{i}:␣{np.round(model_error_dict_forward[i],␣3)}␣m␣({np.round
            (model_error_dict_forward[i]/(1000),␣2)}␣km,␣{np.round(model_error_dict_forward[i
            ]/(0.01*constants.ASTRONOMICAL_UNIT),␣8)}%)")

    if backward:
        system_state_array_backward = model.subsequent_runs(direction='backward')
        model_error_dict_backward = validation.get_accuracy(system_state_array_backward,
            JPL_state_Earth, types=['last', 'max'])
        for i in model_error_dict_backward:
            print(f"distance␣backward␣{i}:␣{np.round(model_error_dict_backward[i],␣3)}␣m␣({np
                .round(model_error_dict_backward[i]/(1000),␣2)}␣km,␣{np.round(
                model_error_dict_backward[i]/(0.01*constants.ASTRONOMICAL_UNIT),␣8)}%)")

def run_once():
    print('model␣start')
    # validation.JPL_Sun_Earth(JPL_state_Earth)

    system_state_array = model.run()
    print('Model␣done\n')
    model_error_dict = validation.get_accuracy(system_state_array, JPL_state_Earth, types=['
        last', 'max'])
    for i in model_error_dict:
        print(f"distance␣{i}:␣{np.round(model_error_dict[i],␣3)}␣m␣({np.round(
            model_error_dict[i]/(1000),␣2)}␣km,␣{np.round(model_error_dict[i]/(0.01*constants
            .ASTRONOMICAL_UNIT),␣8)}%)")

    validation.JPL(system_state_array, JPL_state_Earth)

def check_integrators():
    # code for running multiple integrators

    # start lists for plotting
    integrator_index_range = 2 # 8
    settings_index_range = 6 # 6
    results_arr = np.zeros((integrator_index_range, settings_index_range, 2))

    print('models␣start')
    for integrator_index in range(integrator_index_range):
        integrator_index_USED = integrator_index + 3
        for settings_index in range(settings_index_range):
            # integrator_index = integrator_index
            settings_index_USED = settings_index
            stepsize = stepsizes[settings_index_USED]
            print(f"{integrator_name_lst[integrator_index_USED]},␣step␣{stepsize}s")
            model_start = time.time()
            system_state_array, dep_vars_array = model.run(integrator_index=
                integrator_index_USED, settings_index=settings_index_USED)
            model_end = time.time()

            model_time = model_end - model_start
            model_error = validation.get_accuracy(system_state_array, JPL_state_Earth)
            results_arr[integrator_index, settings_index, 0] = model_time
            results_arr[integrator_index, settings_index, 1] = model_error

    plot.integrators(results_arr)

def check_propagators():
    # code for running multiple propagators

    # start lists for plotting
    propagator_index_range = 2 # 7
    settings_index_range = 1 # 2
    results_arr = np.zeros((propagator_index_range, settings_index_range, 2))

    print('models␣start')
    for propagator_index in range(propagator_index_range):
        propagator_index_USED = propagator_index
        for settings_index in range(settings_index_range):
            settings_index_USED = settings_index
            stepsize = stepsizes[settings_index_USED]
            print(f"{propagator_name_lst[propagator_index_USED]},␣step␣{stepsize}s")

            system_state_array, model_time = model.run(propagator_index=propagator_index_USED
                ,
                                                       settings_index=settings_index_USED,
                                                       timer=True)

            model_error = validation.get_accuracy(system_state_array, JPL_state_Earth)
            results_arr[propagator_index, settings_index, 0] = model_time
            results_arr[propagator_index, settings_index, 1] = model_error

    plot.integrators(results_arr)

def check_Spherical_Harmonics():
    # code for checking the influence of the spherical harmonics on the model
    print('models␣start')
    models_analyzed = [0, 3, 10, 16]
    # models_analyzed = np.linspace(0, 2, num=3, dtype=int)
    results = dict()

    for model_number in models_analyzed:
        time_begin = time.time()
        system_state_array = model.subsequent_runs(model_number=model_number)
```

```python
        CPU_time = time.time() - time_begin
        model_error_dict = validation.get_accuracy(system_state_array, JPL_state_Earth, types=['last', 'max'])
        results[model_number] = model_error_dict['last'], model_error_dict['max'], CPU_time
    plot.SH(results)

def processing_to_graph():
    model_chosen = 'model0_2500y'
    filename_forward = f'{model_chosen}_forward_lim'
    filename_backward = f'{model_chosen}_backward_lim'
    data_array_forward = utilities.read_file(filename_forward)
    data_array_backward = utilities.read_file(filename_backward)
    print("finished␣reading,␣start␣processing")

    # ||Earth - Sun||
    diff_forward = np.zeros((np.shape(data_array_forward[:, :5])))
    diff_forward[:, 1:4] = data_array_forward[:, 1:4] - data_array_forward[:, 13:16]
    diff_forward[:, 0] = data_array_forward[:, 0]

    diff_backward = np.zeros((np.shape(data_array_backward[:, :5])))
    diff_backward[:, 1:4] = data_array_backward[:, 1:4] - data_array_backward[:, 13:16]
    diff_backward[:, 0] = data_array_backward[:, 0]

    # ||Earth|| - ||Sun||
    diff_forward[:, 4] = np.linalg.norm(data_array_forward[:, 1:4], axis=1) - np.linalg.norm(data_array_forward[:, 13:16], axis=1)
    diff_backward[:, 4] = np.linalg.norm(data_array_backward[:, 1:4], axis=1) - np.linalg.norm(data_array_backward[:, 13:16], axis=1)

    diff_forward = diff_forward[:-1,:]
    diff_backward = diff_backward[:-1,:]

    # ||Model - JPL|| of Earth wrt SSB
    JPL_forward, JPL_backward = validation.JPL_Sun_Earth(no_plot_return=True)
    JPL_backward = np.flip(JPL_backward, axis=0)

    diff_JPL_forward = np.zeros((np.shape(diff_forward[:, :5])))
    diff_JPL_forward[:, 1:4] = diff_forward[:, 1:4] - JPL_forward[:, 1:]
    diff_JPL_forward[:, 0] = diff_forward[:, 0]
    diff_JPL_forward[:, 4] = np.linalg.norm(diff_forward[:, 1:4], axis=1) - np.linalg.norm(JPL_forward[:, 1:], axis=1)
    assert np.linalg.norm(diff_forward[:, 0] - JPL_forward[:, 0]) == 0

    diff_JPL_backward = np.zeros((np.shape(diff_backward[:, :5])))
    diff_JPL_backward[:, 1:4] = diff_backward[:, 1:4] - JPL_backward[:, 1:]
    diff_JPL_backward[:, 0] = diff_backward[:, 0]
    diff_JPL_backward[:, 4] = np.linalg.norm(diff_backward[:, 1:4], axis=1) - np.linalg.norm(JPL_backward[:, 1:], axis=1)
    assert np.linalg.norm(diff_backward[:, 0] - JPL_backward[:, 0]) == 0

    print("finished␣processing,␣start␣plotting")

    plot.Earth_Sun_Distance(diff_forward, diff_backward, dim3=False)
    # plot.Earth_Sun_Distance(diff_JPL_forward, diff_JPL_backward, alt_title='Earth/SSB distance difference between model and JPL')

def temp_run():
    filename = 'model0_2500y_backward_lim'
    system_state_array_backward = utilities.read_file(filename)
    model_error_dict_backward = validation.get_accuracy(system_state_array_backward, JPL_state_Earth, types=['last', 'max'], mode='backward')
    for i in model_error_dict_backward:
        print(f"distance␣backward␣{i}:␣{np.round(model_error_dict_backward[i],␣3)}␣m␣({np.round(model_error_dict_backward[i]/(1000),␣2)}␣km,␣{np.round(model_error_dict_backward[i]/(0.01*constants.ASTRONOMICAL_UNIT),␣8)}%)")

def generate_sensitivity_models(number_of_runs, margin=20, start=81):
    # margin in km
    # np.random.seed(420)
    import matplotlib.pyplot as plt
    fig = plt.figure()
    ax = fig.add_subplot(projection='3d')
    margin = [margin, margin, margin/1000]

    for i in range(number_of_runs):
        print(f"---------\n␣starting␣run␣{start␣+␣i}␣\n---------")
        sample = np.random.random(3)
        sample = (sample - 0.5) * 2 * margin * 1000
        ax.scatter(sample[0], sample[1], sample[2], c='c')
        # print(start+i)

        model.subsequent_runs(sensitivity = start+i,
                              pertubation = sample)

        model.subsequent_runs(direction = 'backward',
                              sensitivity = start+i,
                              pertubation = sample)

    # plt.show()

def analyize_sensitivity_models(filename, number_of_files):
    # read original model
    baseline_forward = utilities.read_file(filename+'_forward_lim')
    baseline_backward = utilities.read_file(filename+'_backward_lim')
    baseline_0 = baseline_forward[0, 1:4] - baseline_forward[0, 13:16]
    baseline_f = baseline_forward[-1, 1:4] - baseline_forward[-1, 13:16]
    baseline_b = baseline_backward[-1, 1:4] - baseline_backward[-1, 13:16]

    # read sensitivity
    result = dict()
    for run in range(1, number_of_files+1):
        print(f"loading␣run␣{run}")
        run_forward = utilities.read_file_sensitivity(filename+'_forward_lim', run)
        run_backward = utilities.read_file_sensitivity(filename+'_backward_lim', run)

        # get init diff km wrt end diff km (scatter), total and x,y,z
        run_0 = run_forward[0, 1:4] - run_forward[0, 4:7]
        run_f = run_forward[-1, 1:4] - run_forward[-1, 4:7]
        run_b = run_backward[-1, 1:4] - run_backward[-1, 4:7]


        x = run_0[0] - baseline_0[0], run_f[0] - baseline_f[0], run_b[0] - baseline_b[0]
        y = run_0[1] - baseline_0[1], run_f[1] - baseline_f[1], run_b[1] - baseline_b[1]
        z = run_0[2] - baseline_0[2], run_f[2] - baseline_f[2], run_b[2] - baseline_b[2]
        norm = np.linalg.norm(run_0 - baseline_0), np.linalg.norm(run_f - baseline_f), np.
            linalg.norm(run_b - baseline_b)
        result[run] = x, y, z, norm
    print(f"plotting␣results")
    plot.sensitivity(result)
    # add hemisphere

def init_state_JPL():
    system_state_array_forward = model.subsequent_runs_JPL_state()
    system_state_array_backward = model.subsequent_runs_JPL_state(direction='backward')

def run_5000y():
    # system_state_array = model.subsequent_runs_5000()

    filename_forward = 'model0_2500y_forward_lim'
    filename_backward = 'model0_2500y_backward_lim'
    data_array_forward = utilities.read_file(filename_forward)
    data_array_backward = utilities.read_file(filename_backward)
    system_state_array = utilities.read_file('model0_2500y_forward_lim', extra=True)
    data_array_backward = data_array_backward[::-1]
    data_array_forward = data_array_forward[1:]
    data_array = np.concatenate((data_array_backward, data_array_forward), axis=0)

    diff = system_state_array - data_array
    plot.plot5000y(system_state_array[:, 0], diff[:, 1:4])

def JPL_init_state_to_graph():
    model_chosen = 'model0_2500y'
    filename_forward = f'{model_chosen}_forward_lim'
    filename_backward = f'{model_chosen}_backward_lim'
    data_model_forward = utilities.read_file(filename_forward)
    data_model_backward = utilities.read_file(filename_backward)
    data_JPL_forward = utilities.read_file_JPL(filename_forward)
    data_JPL_backward = utilities.read_file_JPL(filename_backward)
    print("finished␣reading,␣start␣processing")

    model_forward, model_backward = utilities.get_diffs(data_model_forward,
        data_model_backward)
    JPL_state_forward, JPL_state_backward = utilities.get_diffs(data_JPL_forward,
        data_JPL_backward)

    # ||JPL state - JPL|| of Earth wrt SSB
    JPL_forward, JPL_backward = validation.JPL_Sun_Earth(no_plot_return=True)
    JPL_backward = np.flip(JPL_backward, axis=0)

    diff_JPL_forward = np.zeros((np.shape(JPL_state_forward[:, :5])))
    diff_JPL_forward[:, 1:4] = JPL_state_forward[:, 1:4] - JPL_forward[:, 1:]
    diff_JPL_forward[:, 0] = JPL_state_forward[:, 0]
    diff_JPL_forward[:, 4] = np.linalg.norm(JPL_state_forward[:, 1:4], axis=1) - np.linalg.
        norm(JPL_forward[:, 1:], axis=1)
    assert np.linalg.norm(JPL_state_forward[:, 0] - JPL_forward[:, 0]) == 0

    diff_JPL_backward = np.zeros((np.shape(JPL_state_backward[:, :5])))
    diff_JPL_backward[:, 1:4] = JPL_state_backward[:, 1:4] - JPL_backward[:, 1:]
    diff_JPL_backward[:, 0] = JPL_state_backward[:, 0]
    diff_JPL_backward[:, 4] = np.linalg.norm(JPL_state_backward[:, 1:4], axis=1) - np.linalg.
        norm(JPL_backward[:, 1:], axis=1)
    assert np.linalg.norm(JPL_state_backward[:, 0] - JPL_backward[:, 0]) == 0

    diff_model_forward = np.zeros((np.shape(JPL_state_forward[:, :5])))
    diff_model_forward[:, 1:4] = JPL_state_forward[:, 1:4] - model_forward[:, 1:4]
    diff_model_forward[:, 0] = JPL_state_forward[:, 0]
    diff_model_forward[:, 4] = np.linalg.norm(JPL_state_forward[:, 1:4], axis=1) - np.linalg.
        norm(model_forward[:, 1:4], axis=1)
    assert np.linalg.norm(JPL_state_forward[:, 0] - model_forward[:, 0]) == 0

    diff_model_backward = np.zeros((np.shape(JPL_state_backward[:, :5])))
    diff_model_backward[:, 1:4] = JPL_state_backward[:, 1:4] - model_backward[:, 1:4]
    diff_model_backward[:, 0] = JPL_state_backward[:, 0]
    diff_model_backward[:, 4] = np.linalg.norm(JPL_state_backward[:, 1:4], axis=1) - np.
        linalg.norm(model_backward[:, 1:4], axis=1)
    assert np.linalg.norm(JPL_state_backward[:, 0] - model_backward[:, 0]) == 0

    print("finished␣processing,␣start␣plotting")

    # plot.JPL_state(diff_JPL_forward, diff_JPL_backward, diff_model_forward,
        diff_model_backward)
    plot.Earth_Sun_Distance(diff_model_forward, diff_model_backward, alt_title='Earth/SSB␣
        distance␣difference␣between␣model␣(JPL␣state)␣and␣model␣(SPICE␣state)')
    plot.Earth_Sun_Distance(diff_JPL_forward, diff_JPL_backward, alt_title='Earth/SSB␣
        distance␣difference␣between␣model␣(JPL␣state)␣and␣JPL')

def funrun():
    system_state_array, dep_var = model.funrun()
    plot.results(system_state_array)

if __name__ == "__main__":
    # check_Spherical_Harmonics()
    # run_once()
    # long_run(backward=True)
    # long_run(backward=False)
    # processing_to_graph()
    # check_propagators()
    # temp_run()
    # generate_sensitivity_models(20)
    # analyze_sensitivity_models('model0_2500y', 100)
    # run_5000y()
    # init_state_JPL()
    # JPL_init_state_to_graph()
    # check_integrators()
    funrun()
    print("\nDone\n")
```

model.py

```
# Load settings
from settings import *
import utilities

# Load tudatpy modules
from tudatpy.interface import spice
from tudatpy import numerical_simulation
from tudatpy.numerical_simulation import estimation_setup, environment_setup,
    propagation_setup, propagation
from tudatpy import constants
from tudatpy.util import result2array
from tudatpy.astro.time_conversion import DateTime

import time

def run(integrator_index = integrator_setting,
        propagator_index = propagator_setting,
        settings_index = time_step_setting,
        model_number = model_setting,
        system_initial_state = utilities.initial_state(),
        simulation_start = simulation_start_epoch,
        simulation_end = simulation_end_epoch,
        timer = False,
        mod = 1):

    model_start = time.time()
    stepsize = stepsizes[settings_index] * mod
    print(f"{integrator_name_lst[integrator_index]},␣step␣{stepsize}s,␣model␣number␣{
        model_number}")

    # # Load spice kernels
    # spice.load_standard_kernels()

    # Create bodies in simulation.
    bodies_to_propagate = bodies_to_create
    body_settings = environment_setup.get_default_body_settings(bodies_to_create)
    body_system = environment_setup.create_system_of_bodies(body_settings)

    # Central bodies for propagation
    central_bodies = ["SSB"] * len(bodies_to_create)

    # Define the accelerations acting on each body
    acceleration_dict = utilities.get_accelerations(model_number)

    acceleration_models = propagation_setup.create_acceleration_models(
        body_system=body_system,
        selected_acceleration_per_body=acceleration_dict,
        bodies_to_propagate=bodies_to_propagate,
        central_bodies=central_bodies
    )

    # Create termination settings
    termination_settings = propagation_setup.propagator.time_termination(simulation_end)

    # Create numerical integrator settings
    integrator_settings = utilities.get_integrator_settings(integrator_index,
                                                            settings_index,
                                                            simulation_start,
                                                            mod=mod)

    # Create propagation settings
    # dependent_variables_to_save = [estimation_setup.parameter.rotation_pole_position( "
        Earth" )]
```

```python
    dependent_variables_to_save = [propagation_setup.dependent_variable.relative_distance('Earth', 'Sun')]

    propagator_settings = utilities.get_propagator_settings(propagator_index,
                                                            central_bodies,
                                                            acceleration_models,
                                                            bodies_to_propagate,
                                                            system_initial_state,
                                                            simulation_start,
                                                            integrator_settings,
                                                            termination_settings,
                                                            dependent_variables_to_save)

    # Propagate the system of bodies and save the state history (all in one step)
    dynamics_simulator = numerical_simulation.create_dynamics_simulator(
        body_system, propagator_settings)
    results_dep_vars = dynamics_simulator.dependent_variable_history
    results_state = dynamics_simulator.state_history

    # Convert the state dictionnary to a multi-dimensional array
    system_state_array = result2array(results_state)
    dep_vars_array = result2array(results_dep_vars)

    time_taken = time.time() - model_start
    print(f"Model␣finished;␣time␣=␣{np.round(time_taken,␣3)}␣s")
    if timer:
        return system_state_array, dep_vars_array, time_taken
    return system_state_array, dep_vars_array

def subsequent_runs(save = True,
                    model_number = model_setting,
                    direction = 'forward',
                    sensitivity = 0,
                    pertubation = [0, 0, 0]):

    simulation_steps = 5
    simulation_span = simulation_time/simulation_steps * constants.JULIAN_YEAR

    state_array_mid = utilities.initial_state()
    if sensitivity != 0:
        state_array_mid[0:3] = state_array_mid[0:3] + pertubation

    if direction == 'forward':
        mod = 1
    elif direction == 'backward':
        mod = -1

    first = True
    for i in range(simulation_steps):
        simulation_start = simulation_start_epoch + simulation_span * mod * (i)
        simulation_end = simulation_start_epoch + simulation_span * mod * (i+1)

        system_state_array, dep_vars_array = run(simulation_start = simulation_start,
                                                 simulation_end = simulation_end,
                                                 model_number = model_number,
                                                 system_initial_state = state_array_mid,
                                                 mod = mod)

        if direction == 'backward':
            system_state_array = np.flip(system_state_array, axis=0)

        state_array_mid = system_state_array[-1, 1:]

        system_state_array_lim = system_state_array[::974, :]
        if save:
            if sensitivity != 0:
                utilities.save_to_file_sensitivity(system_state_array, name_add=f'p{i}_model{model_number}_{direction}', run=sensitivity)
            else:
                utilities.save_to_file(system_state_array, name_add=f'p{i}_model{model_number}_{direction}')
        del system_state_array
        if first:
            system_state_array_limited = system_state_array_lim
            first = False
        else:
            system_state_array_limited = np.concatenate((system_state_array_limited,
                system_state_array_lim[1:, :]), axis=0)

    if save:
        if sensitivity != 0:
            utilities.save_to_file_sensitivity(system_state_array_limited, name_add=f'
                lim_model{model_number}_{direction}', run=sensitivity)
        else:
            utilities.save_to_file(system_state_array_limited, name_add=f'lim_model{
                model_number}_{direction}')
    return system_state_array_limited

def subsequent_runs_5000(save = True,
                    model_number = model_setting,
                    direction = 'forward',
                    sensitivity = 0,
                    pertubation = [0, 0, 0]):

    simulation_steps = 10
    simulation_time = 5000.
    simulation_start_epoch = -7.889404320000000000e+10
    simulation_span = simulation_time/simulation_steps * constants.JULIAN_YEAR

    state_array_mid = utilities.initial_state(y500run=True)

    if sensitivity != 0:
        state_array_mid[0:3] = state_array_mid[0:3] + pertubation

    if direction == 'forward':
        mod = 1
    elif direction == 'backward':
        mod = -1

    first = True
    for i in range(simulation_steps):
        simulation_start = simulation_start_epoch + simulation_span * mod * (i)
        simulation_end = simulation_start_epoch + simulation_span * mod * (i+1)

        system_state_array, dep_vars_array = run(simulation_start = simulation_start,
                                                 simulation_end = simulation_end,
                                                 model_number = model_number,
                                                 system_initial_state = state_array_mid,
                                                 mod = mod)

        if direction == 'backward':
            system_state_array = np.flip(system_state_array, axis=0)

        state_array_mid = system_state_array[-1, 1:]

        system_state_array_lim = system_state_array[::974, :]
        # if save:
        #     utilities.save_to_file(system_state_array, name_add=f'p{i}_model{model_number}_
            {direction}', folder_add='500y')
        del system_state_array
        if first:
            system_state_array_limited = system_state_array_lim
            first = False
        else:
            system_state_array_limited = np.concatenate((system_state_array_limited,
                system_state_array_lim[1:, :]), axis=0)

    if save:
        utilities.save_to_file(system_state_array_limited, name_add=f'lim_model{model_number}
            _{direction}', folder_add='500y')
    return system_state_array_limited

```

```python
def subsequent_runs_JPL_state(save = True,
                    model_number = model_setting,
                    direction = 'forward',
                    sensitivity = 0,
                    pertubation = [0, 0, 0]):

    simulation_steps = 5
    simulation_span = simulation_time/simulation_steps * constants.JULIAN_YEAR

    state_array_mid = utilities.initial_state_JPL()
    # TO DO: get initial states for all bodies with JPL data!
    if sensitivity != 0:
        state_array_mid[0:3] = state_array_mid[0:3] + pertubation

    if direction == 'forward':
        mod = 1
    elif direction == 'backward':
        mod = -1

    first = True
    for i in range(simulation_steps):
        simulation_start = simulation_start_epoch + simulation_span * mod * (i)
        simulation_end = simulation_start_epoch + simulation_span * mod * (i+1)

        system_state_array, dep_vars_array = run(simulation_start = simulation_start,
                                                  simulation_end = simulation_end,
                                                  model_number = model_number,
                                                  system_initial_state = state_array_mid,
                                                  mod = mod)

        if direction == 'backward':
            system_state_array = np.flip(system_state_array, axis=0)

        state_array_mid = system_state_array[-1, 1:]

        system_state_array_lim = system_state_array[::974, :]
        if save:
            if sensitivity != 0:
                utilities.save_to_file_sensitivity(system_state_array, name_add=f'p{i}_model{
                    model_number}_{direction}_JPL', run=sensitivity)
            else:
                utilities.save_to_file_JPL(system_state_array, name_add=f'p{i}_model{
                    model_number}_{direction}')
        del system_state_array
        if first:
            system_state_array_limited = system_state_array_lim
            first = False
        else:
            system_state_array_limited = np.concatenate((system_state_array_limited,
                system_state_array_lim[1:, :]), axis=0)

    if save:
        if sensitivity != 0:
            utilities.save_to_file_sensitivity(system_state_array_limited, name_add=f'
                lim_model{model_number}_{direction}', run=sensitivity)
        else:
            utilities.save_to_file_JPL(system_state_array_limited, name_add=f'lim_model{
                model_number}_{direction}')
    return system_state_array_limited

def funrun(integrator_index = integrator_setting,
        propagator_index = propagator_setting,
        settings_index = time_step_setting,
        model_number = model_setting,
        system_initial_state = utilities.initial_state(),
        simulation_start = simulation_start_epoch,
        simulation_end = simulation_end_epoch,
        timer = False,
        mod = 1):

    model_start = time.time()
    simulation_end_epoch        = simulation_start_epoch + 1. * constants.JULIAN_YEAR
    stepsize = stepsizes[settings_index] * mod
    print(f"{integrator_name_lst[integrator_index]},␣step␣{stepsize}s,␣model␣number␣{
        model_number}")

    # # Load spice kernels
    # spice.load_standard_kernels()

    # Create bodies in simulation.
    bodies_to_propagate = bodies_to_create
    body_settings = environment_setup.get_default_body_settings(bodies_to_create)
    body_system = environment_setup.create_system_of_bodies(body_settings)

    # Central bodies for propagation
    central_bodies = ["SSB"] * len(bodies_to_create)

    # Define the accelerations acting on each body
    acceleration_dict = utilities.get_accelerations(model_number)

    acceleration_models = propagation_setup.create_acceleration_models(
        body_system=body_system,
        selected_acceleration_per_body=acceleration_dict,
        bodies_to_propagate=bodies_to_propagate,
        central_bodies=central_bodies
    )

    # Create termination settings
    termination_settings = propagation_setup.propagator.time_termination(simulation_end)

    # Create numerical integrator settings
    integrator_settings = utilities.get_integrator_settings(integrator_index,
                                                            settings_index,
                                                            simulation_start,
                                                            mod=mod)

    # Create propagation settings
    # dependent_variables_to_save = [estimation_setup.parameter.rotation_pole_position( "
        Earth" )]
    dependent_variables_to_save = [propagation_setup.dependent_variable.relative_distance('
        Earth', 'Sun')]
    system_initial_state[35] = system_initial_state[35]*1000

    propagator_settings = utilities.get_propagator_settings(propagator_index,
                                                            central_bodies,
                                                            acceleration_models,
                                                            bodies_to_propagate,
                                                            system_initial_state,
                                                            simulation_start,
                                                            integrator_settings,
                                                            termination_settings,
                                                            dependent_variables_to_save)

    # Propagate the system of bodies and save the state history (all in one step)
    dynamics_simulator = numerical_simulation.create_dynamics_simulator(
        body_system, propagator_settings)
    results_dep_vars = dynamics_simulator.dependent_variable_history
    results_state = dynamics_simulator.state_history

    # Convert the state dictionnary to a multi-dimensional array
    system_state_array = result2array(results_state)
    dep_vars_array = result2array(results_dep_vars)

    time_taken = time.time() - model_start
    print(f"Model␣finished;␣time␣=␣{np.round(time_taken,␣3)}␣s")
    if timer:
        return system_state_array, dep_vars_array, time_taken
    return system_state_array, dep_vars_array

if __name__ == "__main__":
    # run()
    print("Done;␣Run␣Model")
```

rotation.py

```python
# from tudatpy.interface import spice
# from tudatpy.numerical_simulation import environment_setup
# from tudatpy import constants

import utilities
import plot
from settings import *

import time
# import numpy as np

spice.load_standard_kernels()
bodies_to_create = ["Earth"]
body_settings = environment_setup.get_default_body_settings(bodies_to_create)
body_system = environment_setup.create_system_of_bodies(body_settings)

# body_fixed_frame_name
body_Earth = body_system.get("Earth")
Earth_rotation_model = body_Earth.rotation_model
earth_rotation_at_epoch = Earth_rotation_model.inertial_to_body_fixed_rotation

def get_rotation_axis(epoch):
    rotation_matrix = earth_rotation_at_epoch(epoch)
    rotation_axis = [0, 0, 1] @ rotation_matrix
    return rotation_axis

def angle_between(v1, v2):
    def unit_vector(vector):
        return vector / np.linalg.norm(vector)

    v1_u = unit_vector(v1)
    v2_u = unit_vector(v2)
    return np.arccos(np.clip(np.dot(v1_u, v2_u), -1.0, 1.0))

def validate_Earth_rotation_angle():
    RM0 = earth_rotation_at_epoch(0)

    Vx = [1, 0, 0] @ RM0
    Vy = [0, 1, 0] @ RM0
    Vz = [0, 0, 1] @ RM0

    Vsun = [0, 0, 1]

    print(f"expect:␣{23.4}␣degrees")
    print(f"x␣{np.rad2deg(angle_between(Vx,␣Vsun))}")
    print(f"y␣{np.rad2deg(angle_between(Vy,␣Vsun))}")
    print(f"z␣{np.rad2deg(angle_between(Vz,␣Vsun))}")

def validate_percentage_formula():
    print(f"deg␣0␣,␣percentage␣{calculate_north_percentage(np.deg2rad(0))},␣=?␣100")
    print(f"deg␣90␣,␣percentage␣{calculate_north_percentage(np.deg2rad(90))},␣=?␣50")
    print(f"deg␣180␣,␣percentage␣{calculate_north_percentage(np.deg2rad(180))},␣=?␣0")
    print(f"deg␣45␣,␣percentage␣{calculate_north_percentage(np.deg2rad(45))},␣=?␣75")
    print(f"deg␣135␣,␣percentage␣{calculate_north_percentage(np.deg2rad(135))},␣=?␣25")

def calculate_north_percentage(theta):
    return (1 - (theta/np.pi)) * 100

def get_Earth_Sun_vector(state):
    Pos_Earth = state[1:4]          # get position Earth
    Pos_Sun = state[13:16]          # get position Sun
    vector = Pos_Sun - Pos_Earth    # get Earth-Sun vector
    return vector

def calc_percentage_north_sun(theta):
    alpha = 2*theta
    R = 6371000
    area_south = ((R*R)/2) * (alpha - np.sin(alpha))
    area_total = np.pi*R*R
    percentage_south = (area_south/area_total) * 100
    percentage_north = 100 - percentage_south
    return percentage_north

def get_sun_percentage_north(state):
    rotation_axis = get_rotation_axis(state[0])
    Earth_Sun_vector = get_Earth_Sun_vector(state)
    angle = angle_between(rotation_axis, Earth_Sun_vector)
    percentage = calc_percentage_north_sun(angle)
    return percentage

def get_SI(state):
    rotation_axis = get_rotation_axis(state[0])
    Earth_Sun_vector = get_Earth_Sun_vector(state)
    angle = angle_between(rotation_axis, Earth_Sun_vector)
    percentage = calc_percentage_north_sun(angle)
    distance = np.linalg.norm(Earth_Sun_vector)
    SI = 1360
    AU = 149598000000
    SI_scaled = (AU/distance)*(AU/distance)*SI
    SI_north = SI_scaled*(percentage/100)
    SI_south = SI_scaled*((100 - percentage)/100)
    return SI_north, SI_south

def get_north_percentage(state):
    rotation_axis = get_rotation_axis(state[0])
    Earth_Sun_vector = get_Earth_Sun_vector(state)
    angle = angle_between(rotation_axis, Earth_Sun_vector)
    percentage = calculate_north_percentage(angle)
    return percentage

def get_order(end, backward=False):
    years = int(end / 974)
    order = [0]
    if backward:
        mod = 1
    else:
        mod = 0
    for i in range(1, (years*12)+1):
        if i%6 != mod:
            order.append(order[-1]+81)
        else:
            order.append(order[-1]+82)
    return order

def calculate_percentages(filename, reduced=True, backward=False):
    time_start = time.time()
    array_lim = utilities.read_file(filename+f'_lim')

    if reduced:
        hemisphere_model_array = np.zeros((((len(array_lim)-1)*12)+1, 2))
        arg_name = 'rotred'
    else:
        hemisphere_model_array = np.zeros((((len(array_lim)-1)*974)+1, 2))
        arg_name = 'rot'


    for p in range(5):
        print(f"Starting␣analysis␣for␣p{p}")
        array = utilities.read_file(filename+f'_p{p}')
        if reduced:
            mod = int(p*12*((len(array_lim)-1)/5)+0.1)
            order = get_order(len(array), backward=backward)
            for i in range(len(order)):
                idx = order[i]
                state = array[idx]
                percentage = get_north_percentage(state)
                hemisphere_model_array[i+mod,:] = state[0], percentage
        else:
            mod = int(p*974*((len(array_lim)-1)/5)+0.1)
            for i in range(len(array)):
                state = array[i]
                percentage = get_north_percentage(state)
                hemisphere_model_array[i+mod,:] = state[0], percentage

    utilities.save_to_file(hemisphere_model_array, filename=filename, name_add=arg_name)
    print(f"time␣=␣{time.time()␣-␣time_start}")

def calculate_percentages_sun(filename, reduced=True, backward=False):
    time_start = time.time()
    array_lim = utilities.read_file(filename+f'_lim')

    if reduced:
        hemisphere_model_array = np.zeros((((len(array_lim)-1)*12)+1, 2))
        arg_name = 'sunrotred'
    else:
        hemisphere_model_array = np.zeros((((len(array_lim)-1)*974)+1, 2))
        arg_name = 'sunrot'


    for p in range(5):
        print(f"Starting␣analysis␣for␣p{p}")
        array = utilities.read_file(filename+f'_p{p}')
        if reduced:
            mod = int(p*12*((len(array_lim)-1)/5)+0.1)
            order = get_order(len(array), backward=backward)
            for i in range(len(order)):
                idx = order[i]
                state = array[idx]
                percentage = get_sun_percentage_north(state)
                hemisphere_model_array[i+mod,:] = state[0], percentage
        else:
            mod = int(p*974*((len(array_lim)-1)/5)+0.1)
            for i in range(len(array)):
                state = array[i]
                percentage = get_sun_percentage_north(state)
                hemisphere_model_array[i+mod,:] = state[0], percentage
    print(arg_name)
    utilities.save_to_file(hemisphere_model_array, filename=filename, name_add=arg_name)
    print(f"time␣=␣{time.time()␣-␣time_start}")

def calculate_SI(filename, reduced=True, backward=False):
    time_start = time.time()
    array_lim = utilities.read_file(filename+f'_lim')

    if reduced:
        hemisphere_model_array = np.zeros((((len(array_lim)-1)*12)+1, 3))
        arg_name = 'SI'
    else:
        hemisphere_model_array = np.zeros((((len(array_lim)-1)*974)+1, 3))
        arg_name = 'SI'


    for p in range(5):
        print(f"Starting␣analysis␣for␣p{p}")
        array = utilities.read_file(filename+f'_p{p}')
        if reduced:
            mod = int(p*12*((len(array_lim)-1)/5)+0.1)
            order = get_order(len(array), backward=backward)
            for i in range(len(order)):
                idx = order[i]
                state = array[idx]
                SI_north, SI_south = get_SI(state)
                hemisphere_model_array[i+mod,:] = state[0], SI_north, SI_south
        else:
            mod = int(p*974*((len(array_lim)-1)/5)+0.1)
            for i in range(len(array)):
                state = array[i]
                percentage = get_SI(state)
                hemisphere_model_array[i+mod,:] = state[0], percentage
```

```python
    print(arg_name)
    utilities.save_to_file(hemisphere_model_array, filename=filename, name_add=arg_name)
    print(f"time␣=␣{time.time()␣-␣time_start}")

def calculate_nondirectional(filename, reduced=True):
    calculate_percentages(filename+'_forward', reduced=reduced)
    calculate_percentages(filename+'_backward', reduced=reduced, backward=True)

def calculate_sun_nondirectional(filename, reduced=True):
    calculate_percentages_sun(filename+'_forward', reduced=reduced)
    calculate_percentages_sun(filename+'_backward', reduced=reduced, backward=True)

def calculate_SI_nondirectional(filename, reduced=True):
    calculate_SI(filename+'_forward', reduced=reduced)
    calculate_SI(filename+'_backward', reduced=reduced, backward=True)

def process_hemisphere(filename, reduced=True, sun=False):
    if sun:
        arg = 'sunrot'
    else:
        arg = 'rot'
    if reduced:
        array_f = utilities.read_file(filename+f'_forward_{arg}red')
        array_b = utilities.read_file(filename+f'_backward_{arg}red')

    years = int(((len(array_f)-1)/12)+0.1)
    data = np.zeros((2, 18, years))
    array_f = array_f[:-1]
    array_b = array_b[::-1]
    array_b = array_b[:-1]

    for i in range(len(array_f)):
        a = 6 + i%12
        b = 0 + int(i/12)
        data[0, a, b] = array_f[i][1]
        data[1, a, b] = array_b[i][1]

    for year in range(years):
        data[0, 0, year] = int((array_f[year*12][0]-simulation_start_epoch)/constants.
            JULIAN_YEAR + 2000.1)
        data[1, 0, year] = int((array_b[year*12][0]-simulation_start_epoch)/constants.
            JULIAN_YEAR + 2000.1)

        data[0, 2, year] = np.average(data[0, 6:9, year])
        data[0, 3, year] = np.average(data[0, 9:12, year])
        data[0, 4, year] = np.average(data[0, 12:15, year])
        data[0, 5, year] = np.average(data[0, 15:18, year])
        data[0, 1, year] = np.average(data[0, 2:6, year])

        data[1, 2, year] = np.average(data[1, 6:9, year])
        data[1, 3, year] = np.average(data[1, 9:12, year])
        data[1, 4, year] = np.average(data[1, 12:15, year])
        data[1, 5, year] = np.average(data[1, 15:18, year])
        data[1, 1, year] = np.average(data[1, 2:6, year])

    plot.hemis_report(data)

def process_SI(filename):
    arg = 'SI'
    array_f = utilities.read_file(filename+f'_forward_{arg}')
    array_b = utilities.read_file(filename+f'_backward_{arg}')

    years = int(((len(array_f)-1)/12)+0.1)
    data = np.zeros((2, 18, years, 2))
    array_f = array_f[:-1]
    array_b = array_b[::-1]
    array_b = array_b[:-1]

    for i in range(len(array_f)):
        a = 6 + i%12
        b = 0 + int(i/12)
        data[0, a, b, :] = array_f[i][1:]
        data[1, a, b, :] = array_b[i][1:]

    for year in range(years):
        data[0, 0, year, :] = int((array_f[year*12][0]-simulation_start_epoch)/constants.
            JULIAN_YEAR + 2000.1)
        data[1, 0, year, :] = int((array_b[year*12][0]-simulation_start_epoch)/constants.
            JULIAN_YEAR + 2000.1)

        data[0, 2, year, :] = np.average(data[0, 6:9, year, :], axis=0)
        data[0, 3, year, :] = np.average(data[0, 9:12, year, :], axis=0)
        data[0, 4, year, :] = np.average(data[0, 12:15, year, :], axis=0)
        data[0, 5, year, :] = np.average(data[0, 15:18, year, :], axis=0)
        data[0, 1, year, :] = np.average(data[0, 2:6, year, :], axis=0)

        data[1, 2, year, :] = np.average(data[1, 6:9, year, :], axis=0)
        data[1, 3, year, :] = np.average(data[1, 9:12, year, :], axis=0)
        data[1, 4, year, :] = np.average(data[1, 12:15, year, :], axis=0)
        data[1, 5, year, :] = np.average(data[1, 15:18, year, :], axis=0)
        data[1, 1, year, :] = np.average(data[1, 2:6, year, :], axis=0)

    plot.SI_report(data)

if __name__ == "__main__":
    # validate_Earth_rotation_angle()
    # validate_percentage_formula()
    # get_north_percentage(0)
    # print(get_order())
    # calculate_nondirectional('model0_2500y')
    # calculate_percentages('model0_25y_forward')
    # process_hemisphere('model0_2500y', sun=True)
    # calculate_sun_nondirectional('model0_5y')
    # calculate_SI_nondirectional('model0_2500y')
    process_SI('model0_2500y')
    # print(0, calc_percentage_north_sun(np.deg2rad(0)))
    # print(90, calc_percentage_north_sun(np.deg2rad(90)))
    # print(180, calc_percentage_north_sun(np.deg2rad(180)))
    # print(45, calc_percentage_north_sun(np.deg2rad(45)))
    # print(135, calc_percentage_north_sun(np.deg2rad(135)))
    print("Done")
```

settings lists.py

```
integrator_name_lst = ["RK4␣step", "RK4(5)␣step", "RK5(6)␣step", "RK7(8)␣step", "RKDP8(7)␣
    step",
                       "BS4␣step", "BS6␣step", "BS8␣step",
                       "RK4␣tol.", "RK5(6)␣tol.", "RK7(8)␣tol.", "RKDP8(7)␣tol.",
                       "BS4␣tol.", "BS6␣tol.", "BS8␣tol.",
                       "ABM", "ABM", "ABM", "ABM", "ABM", "ABM"]

propagator_name_lst = ['Cowell', 'Gauss␣Kelperian', 'Gauss␣Mod.␣Eq.', 'USM␣Quat.', 'USM␣Mod.␣
    Rod.', 'USM␣Exp.␣Map']

stepsizes = [3600.0, 7200.0, 10800.0, 21600.0, 32400.0, 64800.0]

bodies_to_create_nomoons = [
    "Earth",
    "Moon",
    "Sun",
    "Mercury",
    "Venus",
    "Mars",
    "Jupiter",
    "Saturn",
    "Uranus",
    "Neptune"
]

bodies_to_create_moons = [
    "Earth",    # 1  - 6
    "Moon",     # 7  - 12
    "Sun",      # 13 - 18
    "Mercury",  # 19 - 24
    "Venus",    # 25 - 30
    "Mars",     # 31 - 36
    "Jupiter",  # 37 - 42
    "Saturn",   # 43 - 48
    "Uranus",   # 49 - 54
    "Neptune",  # 55 - 60
    "Io",       # 61 - 66
    "Europa",   # 67 - 72
    "Ganymede", # 73 - 78
    "Callisto", # 79 - 84
    "Titan"     # 85 - 90
]

if __name__ == "__main__":
    print("Done; Run Settings list")
```

settings.py

```
# Load standard modules
import numpy as np
import sys
from settings_lists import *

# Load tudatpy modules
from tudatpy.interface import spice
from tudatpy import numerical_simulation
from tudatpy.numerical_simulation import environment_setup, propagation_setup, propagation
from tudatpy import constants
from tudatpy.util import result2array
from tudatpy.astro.time_conversion import DateTime

spice.load_standard_kernels()

simulation_time              = 1. # years
simulation_start_epoch_JD    = 2451544.50
simulation_start_epoch       = spice.convert_julian_date_to_ephemeris_time(
    simulation_start_epoch_JD)
simulation_end_epoch         = simulation_start_epoch + simulation_time * constants.
    JULIAN_YEAR

# decided standard settings
model_setting       = 0 # point masses
integrator_setting  = 3 # RK7(8) step
time_step_setting   = 4 # 32400.0 s = 9.0 h
propagator_setting  = 0 # Cowell

# Define bodies in simulation
bodies_to_create = bodies_to_create_nomoons

# years define

dates = [[-44179560000.0, -44147937600.0],
        [-28400760000.0, -28369137600.0],
        [-12622824000.0, -12591201600.0],
        [-9467064000.0, -9435528000.0],
        [631108800.0, 662731200.0],
        [18934171200.0, 18965707200.0]]

if __name__ == "__main__":
    print("Done; Run Settings")
```

helper.py

```python
from settings import *
import JPL_data.JPL as JPL

def get_JPL_Earth_Data():
    JPL_dict = JPL.read_JPL(['Earth'])
    JPL_vector_Earth = JPL_dict['Earth'].vector
    JPL_JD = JPL_dict['Earth'].time_JD_bary
    JPL_epoch = (JPL_JD - constants.JULIAN_DAY_ON_J2000) * constants.JULIAN_DAY
    JPL_epoch = np.array([JPL_epoch]).T
    JPL_state_Earth = np.concatenate((JPL_epoch, JPL_vector_Earth), axis=1)
    return JPL_state_Earth

def find_matching_epochs(state_array_1, state_array_2):
    idx_1 = [i for i in range(len(state_array_1[:,0])) if state_array_1[i,0] in state_array_2
        [:,0]]
    idx_2 = [i for i in range(len(state_array_2[:,0])) if state_array_2[i,0] in state_array_1
        [:,0]]
    return idx_1, idx_2

def initial_state():
    spice.load_standard_kernels()
    bodies_to_propagate = bodies_to_create
    body_settings = environment_setup.get_default_body_settings(bodies_to_create)
    body_system = environment_setup.create_system_of_bodies(body_settings)
    central_bodies = ["SSB"] * len(bodies_to_create)

    system_initial_state = propagation.get_initial_state_of_bodies(
        bodies_to_propagate=bodies_to_propagate,
        central_bodies=central_bodies,
        body_system=body_system,
        initial_time=simulation_start_epoch
    )

    return system_initial_state
```

utilities.py

```python
from settings import *
import utilities
import JPL_data.JPL as JPL
from pathlib import Path

def get_JPL_Earth_Data(alt=False):
    JPL_dict = JPL.read_JPL(['Earth'], alt=alt)
    JPL_vector_Earth = JPL_dict['Earth'].vector
    JPL_JD = JPL_dict['Earth'].time_JD_bary
    JPL_epoch = (JPL_JD - constants.JULIAN_DAY_ON_J2000) * constants.JULIAN_DAY
    JPL_epoch = np.array([JPL_epoch]).T
    JPL_state_Earth = np.concatenate((JPL_epoch, JPL_vector_Earth), axis=1)
    return JPL_state_Earth

def get_JPL_Data(body_name, alt=False):
    if type(body_name) is list:
        body_list = body_name
    else:
        body_list = [body_name]

    for body in body_list:
        JPL_dict = JPL.read_JPL([body], alt=alt)
        if body_list[0] == body:
            JPL_JD = JPL_dict[body].time_JD_bary
            JPL_epoch = (JPL_JD - constants.JULIAN_DAY_ON_J2000) * constants.JULIAN_DAY
            JPL_state = np.array([JPL_epoch]).T
        JPL_vector = JPL_dict[body].vector
        JPL_state = np.concatenate((JPL_state, JPL_vector), axis=1)
    return JPL_state

def find_idx_epoch(JPL_state_array, epoch=simulation_start_epoch):
    [idx] = np.where(JPL_state_array[:,0] == epoch)[0]
    return idx

def find_matching_epochs(state_array_1, state_array_2):
    idx_1 = [i for i in range(len(state_array_1[:,0])) if state_array_1[i,0] in state_array_2[:,0]]
    idx_2 = [i for i in range(len(state_array_2[:,0])) if state_array_2[i,0] in state_array_1[:,0]]
    return idx_1, idx_2

def find_matching_epochs_JPL(state_array_1, JPL_state_array, mode):
    # JPL_state_array_orig = JPL_state_array
    if mode == 'forward':
        idx_start = utilities.find_idx_epoch(JPL_state_array, simulation_start_epoch)
        JPL_state_array = JPL_state_array[idx_start:, :]
        modifier = 0
    elif mode == 'backward':
        idx_start = utilities.find_idx_epoch(JPL_state_array, simulation_start_epoch)
        JPL_state_array = JPL_state_array[:idx_start+1, :]
        modifier = 1
    else:
        raise "incorrect␣input␣@find_matching_epochs_JPL"

    idx_lst = []
    idx_JPL = []
    for i in range(len(JPL_state_array[:,0])):
        # print(np.shape(state_array_1))
        # print(JPL_state_array[i,0])
        idx = np.where(state_array_1[:,0] == JPL_state_array[i,0])[0]
        if len(idx) > 0:
            idx_lst.append(idx)
            idx_JPL.append(i + idx_start - modifier*(len(JPL_state_array[:,0]) - 1))

    if mode == 'backward':
        idx_lst = np.flip(idx_lst)
        idx_JPL = np.flip(idx_JPL)

    # for i in range(len(idx_lst)):
    #     print(i, idx_lst[i], idx_JPL[i])
    #     print(i, state_array_1[idx_lst[i], 0], JPL_state_array_orig[idx_JPL[i], 0])
    return idx_lst, idx_JPL

    # idx_1 = [i for i in range(len(state_array_1[:,0])) if state_array_1[i,0] in state_array_2[:,0]]
    # idx_2 = [i for i in range(len(state_array_2[:,0])) if state_array_2[i,0] in state_array_1[:,0]]
    # return idx_1, idx_2

def initial_state(y500run=False):
    if y500run:
        state_str_mid_2500 = '-6.470562227385867310e+10␣1.310027482415024109e+11␣
            7.608546149423669577e+08␣-2.720092960963146470e+04␣-1.324227245512727859e+04␣
            -7.352591003911769008e+01␣-6.486121674887060547e+10␣1.313758373315393219e+11␣
            7.496277058963145018e+08␣-2.809166671665949980e+04␣-1.363473427448780603e+04␣
            -1.583862746844457376e+02␣5.335421729967144728e+08␣-8.672159074094423056e+08␣
            -8.486368156289873645e+06␣1.300396154336692511e+01␣7.232047489014981956e+00␣
            -3.710268718474826755e-01␣-3.441117190070409393e+10␣-6.064560993250991821e+10␣
            -1.259038468100996017e+09␣3.239611786292151010e+04␣-2.163711527795833899e+04␣
            -4.871661222290700607e+03␣1.223932267319582748e+10␣-1.091500405479745483e+11␣
            -1.421208496675765753e+09␣3.455809041614567832e+04␣3.607018176139844400e+03␣
            -2.013111400722681083e+03␣3.992923985302412415e+10␣-2.092586720573141174e+11␣
            -5.269070456498918533e+09␣2.508408839018704020e+04␣6.328660687718056579e+03␣
            -6.213493438631658137e+02␣-4.637087509374016113e+11␣6.411027441350236816e+11␣
            9.218116495443857193e+09␣-1.067627721619303702e+04␣-7.086044214174676199e+03␣
            2.737386346564274504e+02␣-2.687098494056245117e+11␣1.314764422913289795e+12␣
            -1.764406901498619461e+10␣-1.007123762843288387e+04␣-1.805307647262527553e+03␣
            4.021475134004135157e+02␣2.292395473419145508e+12␣1.875286986598077881e+12␣
            -2.312062281263795471e+10␣-4.350524003897318835e+03␣4.952543419808044746e+03␣
            8.045428094245463058e+01␣-2.170905792319677490e+12␣-3.983706911111201660e+12␣
            1.317386975560655975e+11␣4.733091686834714892e+03␣-2.570216081103309079e+03␣
            -5.560726238849181868e+01␣-4.643014826422095337e+11␣6.412070313934803467e+11␣
            9.260335034823532104e+09␣-1.335151152362332505e+04␣-2.142001934373380936e+04␣
            4.007650100211997142e+02␣-4.630394307882062988e+11␣6.408383359417388916e+11␣
            9.161710621372217178e+09␣-5.579605499613249776e+03␣5.310678374673068902e+03␣
            -6.880027017964873437e+01␣-4.638095220154948730e+11␣6.421657696064432373e+11␣
            9.236319202002666473e+09␣-2.150632848559605554e+04␣-8.157269520155009559e+03␣
            1.063787164693600516e+03␣-4.650795536571047974e+11␣6.398092690352264404e+11␣
            9.312740507256805420e+09␣-5.180070288234085638e+03␣-1.313551744352822971e+04␣
            -1.501505318592949436e+02␣-2.698488563181660156e+11␣1.314285920748873535e+12␣
            -1.760226219728828430e+10␣-7.907103930582629800e+03␣-6.243306499910396269e+03␣
            2.842703874601901589e+03'
        # state_str_mid_5 = '-2.545190456064811325e+10 1.460153806213573303e+11
            1.866423953382877167e+06 -2.985521394114493523e+04 -5.232966840167215196e+03
            -6.267535224009377348e-01 -2.545127962679296494e+10 1.456551176810104370e+11
            2.523407612905007601e+07 -2.876609246282456297e+04 -5.242441148169854387e+03
            6.673307745669779933e+01 -1.442548564976629913e+08 1.104860533722047806e+09
            8.317804309368794784e+05 -1.151808436050178486e+01 3.112656478142366012e+00
            2.909095203747575353e-01 4.359892600132970428e+10 -4.265005303920007324e+10
            -7.588788801630222321e+09 2.478271216354580247e+04 3.676008165559506597e+04
            7.266057346508196133e+02 -7.643140550959671021e+10 7.662048990445036316e+10
            5.435827197388555527e+09 -2.477561268087668941e+04 -2.506958810635399277e+04
            1.087492090142828829e+03 -1.379170670441119995e+11 2.040474817976522217e+11
            7.639850361652002335e+09 -1.914073678275265047e+04 -1.154465440698802558e+04
            2.288083651757158918e+02 -4.181024886034146729e+11 -6.867226326978796387e+11
            1.220782932808006668e+10 1.099207467255302436e+04 -6.180866989292461767e+03
            -2.204815738611930271e+02 1.384117579045961182e+12 -4.223142080745545654e+11
            -4.766555335347930908e+10 2.295597938315998817e+03 9.218478728204845538e+03
            -2.516125644009802045e+02 1.305081970851829346e+12 -2.639579139656285156e+12
            -2.671523274377833939e+10 6.054150538519555084e+03 2.702140799906237589e+03
            -6.841457871141253122e+01 1.766617730362173584e+12 -4.152436750748621094e+12
            4.479726926479429626e+10 4.966299434453502727e+03 2.158418655852421580e+03
            -1.588895620715615564e+02 -4.185960305037707520e+11 -6.866132667971370850e+11
            1.220478695517108727e+10 7.633630547216225750e+03 -2.181185580698964986e+04
            -8.235990739026721030e+02 -4.176307918522187500e+11 -6.863174651060482178e+11
            1.222763832501715088e+10 1.845809420390800597e+03 5.517752019272636971e+03
            -4.518049570597617048e+01 -4.189515795053150635e+11 -6.860677648125463867e+11
            1.222289916453612709e+10 4.388809853828039195e+03 -1.481191344933412074e+04
            -6.089035498397576021e+02 -4.179680288302916870e+11 -6.885961754358083496e+11
            1.214785215705461693e+10 1.918353346250520190e+04 -5.539115482312001404e+03
            -9.228747341413533434e+01 1.384712883021259033e+12 -4.232551425454268799e+11
            -4.723778968187474823e+10 7.169897697027721733e+03 1.163367778127543534e+04
            -1.976657107441566495e+03'
        state_array_mid = state_str_mid_2500.split('␣')
        # state_array_mid = state_str_mid_5.split(' ')

        state_array_new = np.zeros(len(state_array_mid))
        for i in range(len(state_array_mid)):
            state_array_new[i] = float(state_array_mid[i])
        system_initial_state = np.array(state_array_mid)

        return system_initial_state

    bodies_to_propagate = bodies_to_create
    body_settings = environment_setup.get_default_body_settings(bodies_to_create)
    body_system = environment_setup.create_system_of_bodies(body_settings)
    central_bodies = ["SSB"] * len(bodies_to_create)

    system_initial_state = propagation.get_initial_state_of_bodies(
        bodies_to_propagate=bodies_to_propagate,
        central_bodies=central_bodies,
        body_system=body_system,
        initial_time=simulation_start_epoch
    )

    return system_initial_state

def get_integrator_settings(integrator_index: int,
                            settings_index: int,
                            simulation_start_epoch: float,
                            mod=1):
    """
    Retrieves the integrator settings.
    Function adapted from lecture Propagation and Optimisation.
```

```python

    Parameters
    ----------
    integrator_index : int
        Index that selects the integrator type
    settings_index : int
        Index that selects the tolerance or the step size (depending on the integrator type).
    simulation_start_epoch : float
        Start of the simulation [s] with t=0 at J2000.

    Returns
    -------
    integrator_settings : tudatpy.kernel.numerical_simulation.propagation_setup.integrator.IntegratorSettings
        Integrator settings to be provided to the dynamics simulator.
    """
    # Define list of multi-stage integrators
    integrator = propagation_setup.integrator
    multi_stage_integrators = [integrator.CoefficientSets.rkf_45,
                               integrator.CoefficientSets.rkf_56,
                               integrator.CoefficientSets.rkf_78,
                               integrator.CoefficientSets.rkdp_87]
    fixed_step_size = stepsizes[settings_index] * mod
    step_size_validation = integrator.step_size_validation(minimum_step=fixed_step_size,
                                                           maximum_step=fixed_step_size)
    step_size_control = integrator.step_size_control_elementwise_scalar_tolerance(relative_error_tolerance=np.inf,
                                                                                  absolute_error_tolerance=np.inf)

    # RK4, fixed step-size
    if integrator_index == 0:
        # Create integrator settings
        integrator_settings = integrator.runge_kutta_4(simulation_start_epoch,
                                                       fixed_step_size)

    # RK, fixed step-size
    elif 1 <= integrator_index < 5:
        # Select variable-step integrator
        current_coefficient_set = multi_stage_integrators[integrator_index-1]
        # Create integrator settings
        integrator_settings = integrator.runge_kutta_variable_step(fixed_step_size,
                                                                   current_coefficient_set,
                                                                   step_size_control,
                                                                   step_size_validation)
    # BS, fixed step-size
    elif 5 <= integrator_index < 8:
        extrapol_seq = integrator.bulirsch_stoer_sequence
        BS_num = int(integrator_index * 2.0 - 6.0)
        integrator_settings = integrator.bulirsch_stoer_variable_step(fixed_step_size,
                                                                      extrapol_seq,
                                                                      BS_num,
                                                                      step_size_control,
                                                                      step_size_validation)

    # RK, variable step-size
    elif 8 <= integrator_index < 12:
        # Select variable-step integrator
        current_coefficient_set = multi_stage_integrators[integrator_index-8]
        # Compute current tolerance
        current_tolerance = 10.0 ** (-15.0 + settings_index)
        # Here (epsilon, inf) are set as respectively min and max step sizes
        # also note that the relative and absolute tolerances are the same value
        integrator_settings = integrator.runge_kutta_variable_step_size(simulation_start_epoch,
                                                                        1.0,
                                                                        current_coefficient_set,
                                                                        np.finfo(float).eps,
                                                                        np.inf,
                                                                        current_tolerance,
                                                                        current_tolerance)

    # BS, variable step-size
    elif 12 <= integrator_index < 15:
        current_tolerance = 10.0 ** (-15.0 + settings_index)
        extrapol_seq = integrator.bulirsch_stoer_sequence
        BS_num = int(integrator_index * 2.0 - 6.0)
        integrator_settings = integrator.bulirsch_stoer_variable_step(simulation_start_epoch,
                                                                      1.0,
                                                                      extrapol_seq,
                                                                      BS_num,
                                                                      np.finfo(float).eps,
                                                                      np.inf,
                                                                      current_tolerance,
                                                                      current_tolerance)

    # ABM, variable step-size
    elif integrator_index >= 15:
        current_tolerance = 10.0 ** (-13.0 + settings_index)
        orders = [[6,10],[6,11],[6,12],[7,10],[7,11],[7,12]]
        order_used = orders[integrator_index-15]
        print(order_used)

        integrator_settings = integrator.adams_bashforth_moulton(simulation_start_epoch,
                                                                 1.0,
                                                                 np.finfo(float).eps,
                                                                 np.inf,
                                                                 current_tolerance,
                                                                 current_tolerance,
                                                                 order_used[0],
                                                                 order_used[1])

    return integrator_settings

def get_propagator_settings(propagator_index: int,
                            central_bodies,
                            acceleration_models,
                            bodies_to_propagate,
                            system_initial_state,
                            simulation_start,
                            integrator_settings,
                            termination_settings,
                            output_variables):

    propagators = [propagation_setup.propagator.cowell
                #    propagation_setup.propagator.encke,
                #    propagation_setup.propagator.gauss_keplerian,
                #    propagation_setup.propagator.gauss_modified_equinoctial,
                #    propagation_setup.propagator.unified_state_model_quaternions,
                #    propagation_setup.propagator.
                    unified_state_model_modified_rodrigues_parameters,
                #    propagation_setup.propagator.unified_state_model_exponential_map
                   ]

    current_propagator = propagators[propagator_index]
    propagation_settings = propagation_setup.propagator.translational(
        central_bodies,
        acceleration_models,
        bodies_to_propagate,
        system_initial_state,
        simulation_start,
        integrator_settings,
        termination_settings,
        propagator = current_propagator,
        output_variables = output_variables
    )

    return propagation_settings

def get_accelerations(model_choice=0):
    acceleration_dict = {}
    for body_i in bodies_to_create:
        current_accelerations = {}
        for body_j in bodies_to_create:
            if body_i != body_j:
                accel = propagation_setup.acceleration
                if model_choice == 1 and body_j == 'Mercury':
                    current_accelerations[body_j] = [accel.spherical_harmonic_gravity(2,2)]
                elif model_choice == 2 and body_j == 'Venus':
                    current_accelerations[body_j] = [accel.spherical_harmonic_gravity(2,2)]
                elif model_choice == 3 and body_j == 'Earth':
                    current_accelerations[body_j] = [accel.spherical_harmonic_gravity(2,2)]
                elif model_choice == 4 and body_j == 'Earth':
                    current_accelerations[body_j] = [accel.spherical_harmonic_gravity(4,4)]
                elif model_choice == 5 and body_j == 'Earth':
                    current_accelerations[body_j] = [accel.spherical_harmonic_gravity(8,8)]
                elif model_choice == 6 and body_j == 'Moon':
                    current_accelerations[body_j] = [accel.spherical_harmonic_gravity(2,2)]
                elif model_choice == 7 and body_j == 'Moon':
                    current_accelerations[body_j] = [accel.spherical_harmonic_gravity(4,4)]
                elif model_choice == 8 and body_j == 'Moon':
                    current_accelerations[body_j] = [accel.spherical_harmonic_gravity(8,8)]
                elif model_choice == 9 and body_j == 'Mars':
                    current_accelerations[body_j] = [accel.spherical_harmonic_gravity(2,2)]
                elif model_choice == 10 and body_j == 'Jupiter':
                    current_accelerations[body_j] = [accel.spherical_harmonic_gravity(2,2)]
                elif model_choice == 11 and body_j == 'Jupiter':
                    current_accelerations[body_j] = [accel.spherical_harmonic_gravity(4,4)]
                elif model_choice == 12 and body_j == 'Io':
                    current_accelerations[body_j] = [accel.spherical_harmonic_gravity(2,2)]
                elif model_choice == 13 and body_j == 'Europa':
                    current_accelerations[body_j] = [accel.spherical_harmonic_gravity(2,2)]
                elif model_choice == 14 and body_j == 'Ganymede':
                    current_accelerations[body_j] = [accel.spherical_harmonic_gravity(2,2)]
                elif model_choice == 15 and body_j == 'Callisto':
                    current_accelerations[body_j] = [accel.spherical_harmonic_gravity(2,2)]
                elif model_choice == 16 and body_j == 'Earth':
                    current_accelerations[body_j] = [accel.spherical_harmonic_gravity(2,2)]
                elif model_choice == 16 and body_j == 'Jupiter':
                    current_accelerations[body_j] = [accel.spherical_harmonic_gravity(2,2)]
                else:
                    current_accelerations[body_j] = [accel.point_mass_gravity()]
        acceleration_dict[body_i] = current_accelerations

    return acceleration_dict

def save_to_file(data, filename='', name_add='', folder_add=''):
    # determine file name and folder location
    if len(filename) == 0:
        filename_year = f'{int(simulation_time)}y'

    # split name add
    name_given = filename + '_' + name_add
    try:
        name_given = name_given.strip('_')
        test = name_given.split('_')
        for i in test:
            if 'model' in i:
                j = i.strip('model')
                dirname_model = f'model␣{j}'
                filename_model = f'model{j}'
            elif 'ward' in i:
                filename_direction = i
            elif 'p' in i or 'lim' in i or 'rot' in i or 'SI' in i or 'JPL' in i:
                filename_part = i
            elif 'y' in i:
                filename_year = i
    except:
        print('Failed')

    print(filename_part)
```

```python
    # create folder location
    filename = filename_model + '_' + filename_year + '_' + filename_direction + '_' + filename_part
    if len(folder_add) > 0:
        file_location = f'model_data/{folder_add}/{dirname_model}/{filename_year}/{filename_direction}/'
    else:
        file_location = f'model_data/{dirname_model}/{filename_year}/{filename_direction}/'

    Path(f"./{file_location}").mkdir(parents=True, exist_ok=True)
    np.savetxt(f'{file_location}{filename}', data)

def save_to_file_JPL(data, filename='', name_add='', folder_add=''):
    # determine file name and folder location
    if len(filename) == 0:
        filename_year = f'{int(simulation_time)}y'

    # split name add
    name_given = filename + '_' + name_add
    try:
        name_given = name_given.strip('_')
        test = name_given.split('_')
        for i in test:
            if 'model' in i:
                j = i.strip('model')
                dirname_model = f'model␣{j}'
                filename_model = f'model{j}'
            elif 'ward' in i:
                filename_direction = i
            elif 'p' in i or 'lim' in i or 'rot' in i or 'SI' in i:
                filename_part = i
            elif 'y' in i:
                filename_year = i
    except:
        print('Failed')

    print(filename_part)
    # create folder location
    filename = filename_model + '_' + filename_year + '_' + filename_direction + '_' + filename_part
    if len(folder_add) > 0:
        file_location = f'model_data/JPL/{folder_add}/{dirname_model}/{filename_year}/{filename_direction}/'
    else:
        file_location = f'model_data/JPL/{dirname_model}/{filename_year}/{filename_direction}/'

    Path(f"./{file_location}").mkdir(parents=True, exist_ok=True)
    np.savetxt(f'{file_location}{filename}', data)

def save_to_file_sensitivity(data, filename='', name_add='', run=0):
    from pathlib import Path

    data_save = np.zeros((len(data), 7))
    data_save[:, :4] = data[:, :4]
    data_save[:, 4:] = data[:, 13:16]

    # determine file name and folder location
    if len(filename) == 0:
        filename_year = f'{int(simulation_time)}y'

    # split name add
    name_given = filename + '_' + name_add
    try:
        name_given = name_given.strip('_')
        test = name_given.split('_')
        for i in test:
            if 'model' in i:
                j = i.strip('model')
                dirname_model = f'model␣{j}'
                filename_model = f'model{j}'
            elif 'ward' in i:
                filename_direction = i
            elif 'p' in i or 'lim' in i or 'rot' in i or 'rotred' in i:
                filename_part = i
            elif 'y' in i:
                filename_year = i
    except:
        print('Failed')

    # create folder location
    filename = filename_model + '_' + filename_year + '_' + filename_direction + '_' +
        filename_part
    file_location = f'model_data/sensitivity/{dirname_model}/{filename_year}/run␣{run}/{
        filename_direction}/'

    Path(f"./{file_location}").mkdir(parents=True, exist_ok=True)
    np.savetxt(f'{file_location}{filename}', data_save)

def read_file(filename, extra=False):
    names = filename.split("_")
    filename_model = names[0]
    dirname_model = filename_model[:5] + '␣' + filename_model[5]
    filename_year = names[1]
    filename_direction = names[2]

    if extra:
        file_location = f'model_data/500y/{dirname_model}/{filename_year}/{filename_direction
            }/'
    else:
        file_location = f'model_data/{dirname_model}/{filename_year}/{filename_direction}/'
    data = np.loadtxt(f'{file_location}{filename}')
    array = np.array(data)
    return array

def read_file_sensitivity(filename, run):
    names = filename.split("_")
    filename_model = names[0]
    dirname_model = filename_model[:5] + '␣' + filename_model[5]
    filename_year = names[1]
    filename_direction = names[2]
    dirname_run = f'run␣{run}'

    file_location = f'model_data/sensitivity/{dirname_model}/{filename_year}/{dirname_run}/{
        filename_direction}/'
    data = np.loadtxt(f'{file_location}{filename}')
    array = np.array(data)
    return array

def read_file_JPL(filename):
    names = filename.split("_")
    filename_model = names[0]
    dirname_model = filename_model[:5] + '␣' + filename_model[5]
    filename_year = names[1]
    filename_direction = names[2]

    file_location = f'model_data/JPL/{dirname_model}/{filename_year}/{filename_direction}/'
    data = np.loadtxt(f'{file_location}{filename}')
    array = np.array(data)
    return array

def create_limited(filename):
    print("start␣reading")
    data_lst = []
    for i in range(5):
        data_array = utilities.read_file(filename+f'_p{i}')
        data_array_limited = data_array[::974, :]
        del data_array
        data_lst.append(data_array_limited)
        print(f"part␣{i}␣read")

    data_array_lim = np.concatenate((data_lst[0], data_lst[1][1:, :]), axis=0)
    data_array_lim = np.concatenate((data_array_lim, data_lst[2][1:, :]), axis=0)
    data_array_lim = np.concatenate((data_array_lim, data_lst[3][1:, :]), axis=0)
    data_array_lim = np.concatenate((data_array_lim, data_lst[4][1:, :]), axis=0)
    save_to_file(data_array_lim, filename, name_add='_limb')

def initial_state_JPL():
    data = np.loadtxt(f'model_data/JPL/init_state.txt')
    array = np.array(data)
    return array

def create_initial_state_JPL():
    init_state = np.zeros((90))
    for body_id in range(len(bodies_to_create_moons)):
        body = bodies_to_create_moons[body_id]
        print(body)
        JPL_data = get_JPL_Data(body)
        idx_start = find_idx_epoch(JPL_data)
        i = body_id*6
        init_state[i:i+6] = JPL_data[idx_start, 1:]
    Path(f"./model_data/JPL").mkdir(parents=True, exist_ok=True)
    np.savetxt(f'model_data/JPL/init_state.txt', init_state)

def get_diffs(data_array_forward, data_array_backward):
    # ||Earth - Sun||
    diff_forward = np.zeros((np.shape(data_array_forward[:, :5])))
    diff_forward[:, 1:4] = data_array_forward[:, 1:4] - data_array_forward[:, 13:16]
    diff_forward[:, 0] = data_array_forward[:, 0]

    diff_backward = np.zeros((np.shape(data_array_backward[:, :5])))
    diff_backward[:, 1:4] = data_array_backward[:, 1:4] - data_array_backward[:, 13:16]
    diff_backward[:, 0] = data_array_backward[:, 0]

    # ||Earth|| - ||Sun||
    diff_forward[:, 4] = np.linalg.norm(data_array_forward[:, 1:4], axis=1) - np.linalg.norm(
        data_array_forward[:, 13:16], axis=1)
    diff_backward[:, 4] = np.linalg.norm(data_array_backward[:, 1:4], axis=1) - np.linalg.
        norm(data_array_backward[:, 13:16], axis=1)

    diff_forward = diff_forward[:-1,:]
    diff_backward = diff_backward[:-1,:]
    return diff_forward, diff_backward

def process_data_to_month(year_array, year):
    from tudatpy.kernel.astro import time_conversion

    data_array = np.full((6, 100, 2), np.nan)
    if year_array[0, 0] > year_array[1, 0]:
        year_array = year_array[::-1]

    current_month = 1
    data_month = np.full((300, 2), np.nan)
    length = 0
    max_length = 0
    skip = False
    for data_id in range(len(year_array[:, 0])):
        datapoint = year_array[data_id, 0]

        if datapoint in [-44174462400.0, -44174430000.0, -44174397600.0,
                         -28395662400.0, -28395630000.0, -28395597600.0]:
            skip = True

        else:
            try:
                timepoint = time_conversion.date_time_from_epoch(datapoint)
                tp_year = timepoint.year
                tp_month = timepoint.month
                tp_day = timepoint.day - 1
                tp_string = timepoint.iso_string()
                tp_fraction = (timepoint.julian_day() - 0.5)%1
            except:
                print(datapoint)
                timepoint = time_conversion.date_time_from_epoch(datapoint)

        if tp_year != year:
            print("Faulty␣point")
            print(tp_string)
            input('stopped')

        elif current_month != tp_month:
            data_array[current_month-1, :length, :] = data_month[:length, :]
            if max_length < length:
                max_length = length
            del data_month

            current_month = tp_month
            if current_month > 6:
                data_array = data_array[:, :max_length, :]
                return data_array
            data_month = np.zeros((300, 2))
            length = 0

            # print(f'starting month {current_month}')

        if not skip:
            data_month[length, 0] = tp_fraction + tp_day
            data_month[length, 1] = np.linalg.norm(year_array[data_id, 1:4] - year_array[
                data_id, 4:7])
            length = length + 1
        skip = False

        # data_lst.append(
        # counter = counter + 1
        # input()

    return data_array

if __name__ == "__main__":
    # create_limited('model0_5y_forward')
    # create_initial_state_JPL()
    # print(initial_state_JPL())

    # file_location = f'model_data/lastfig/'
    # data = np.loadtxt(f'{file_location}0y')
    # array = np.array(data)
    # process_data_to_month(array, 600)

    print("Done;␣Run␣Utilities")
```