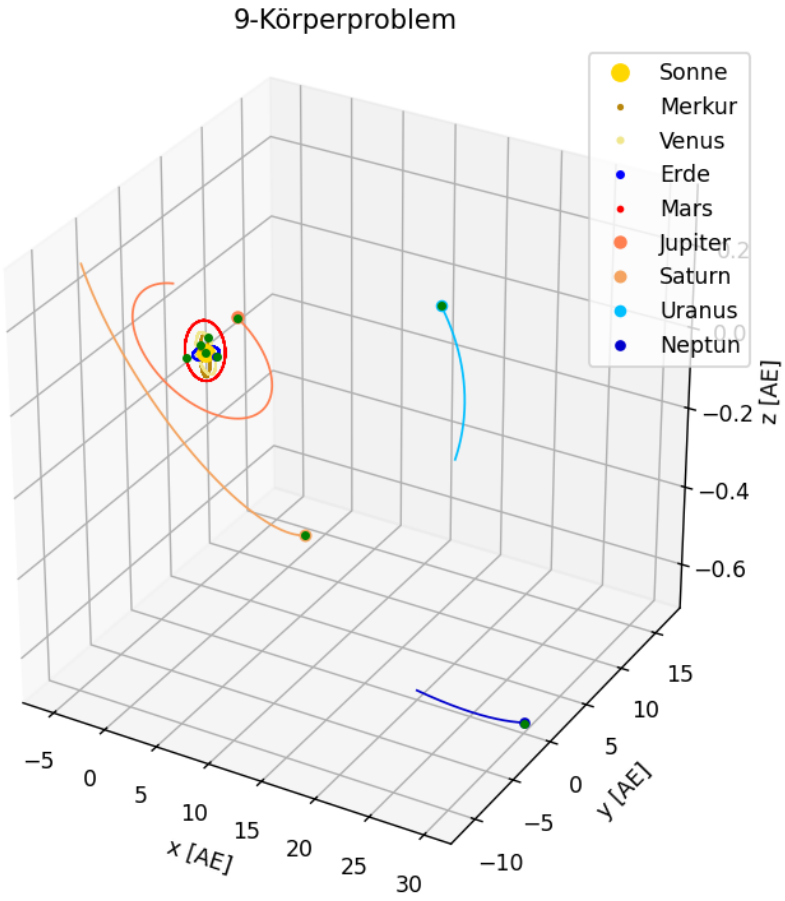
Programmierung einer Simulation des n-Körperproblems und Analyse mit dem Dreikörperproblem und den Planetenorbits

Begleitarbeit zur Maturitätsarbeit



Kantonsschule Zürcher Unterland

Autorin: Michaela Külling

Betreuer: José Abreu Castiñeira
Experte: Manuel Bischof

Maturitätsarbeit 2025/2026

Inhaltsverzeichnis

1 Zusammenfassung

Das Ziel dieser Arbeit ist es, eine Simulation für das n-Körperproblem in Python zu programmieren, welche richtige Resultate liefert und die Grundsätze der Energie- und Impulserhaltung berücksichtigt. Die Beschleunigungen wurden mit dem Newtonschen Gravitationsgesetz berechnet. Relativistische Berechnungen wurden nicht vorgenommen. Für die Berechnungen der Bewegung der Körper wurde der numerische Integrationsalgorithmus *Leapfrog* verwendet. Die Simulation wurde anhand zweier Beispiele getestet. Es wurde der Spezialfall des Dreikörperproblems, bei welchem sich drei Körper der gleichen Masse um ihren gemeinsamen Massenmittelpunkt bewegen, simuliert und analysiert. Ausserdem wurde das Sonnensystem mit der Sonne und allen Planeten über einen Zeitraum von 10 Jahren simuliert. Die Ergebnisse wurden mit den beobachteten Werten verglichen.

2 Einleitung

2.1 Themenfindung

Der Nachthimmel hat mich schon immer fasziniert. Deshalb wusste ich schon bald, dass meine Maturitätsarbeit sich im Bereich der Astronomie bewegen sollte. Beim Nachdenken über mögliche Themen kam ich auf Astrofotografie. Als ich realisierte, in welchem Zeitfenster die Maturitätsarbeit geschrieben werden muss, verwarf ich diesen Gedanken wieder. Die klarsten Nächte für das Beobachten und Fotografieren von Sternen sind nämlich im Januar, Februar und März, also genau die Zeit vom Jahr, in der nicht an der Maturitätsarbeit gearbeitet wird. Da mir das Programmieren im Informatikunterricht sehr gelegen ist, kam mir die Idee, zu einem Phänomen aus der Astrophysik eine Simulation zu programmieren. Zuerst überlegte ich mir die Entstehung und das Ende eines Sternes zu simulieren. Als ich kurz darauf am Science Info Day an der Universität Zürich (UZH) die Vorstellung des Studiengangs Physik besuchte, in welcher der Professor eine Simulation der Kollision von Jupiter mit einem kleineren Gasplaneten zeigte, und dazu sagte, dass es sich bei der Simulation um eine Doktorarbeit handelte, beschloss ich, das Programmieren einer Simulation eines noch relativ unerforschten und physikalisch äusserst komplexen Phänomens wie das der Sternentwicklung Menschen mit mehr physikalischem Wissen und einem Studienabschluss zu überlassen. Ich beschloss ein physikalisch etwas einfacheres und greifbareres Phänomen zu erforschen und eine Simulation dazu zu programmieren. Dabei kam ich auf das Phänomen der Gravitation und der Bewegung der Himmelskörper aufgrund der Massenanziehung. Dieses Phänomen wird mit der Beteiligung von drei Körpern auch das Dreikörperproblem der Himmelsmechanik bezeichnet. Dieses Thema schien nicht allzu unmöglich realisierbar zu sein, da ich im Frühlingssemester 2025 anlässlich des Schüler*innenstudiums an der UZH das Modul *PHY124 Scientific Computing* (vgl. Leistungsausweis im Anhang) bei Prof. Dr. Marcelle Soares dos Santos belegen durfte und ich mir darin viele nützliche Programmiertechniken für wissenschaftliches Programmieren aneignen konnte.

2.2 Zielsetzung

Ziel dieser Arbeit ist es, eine Simulation der Bewegung von verschiedenen Himmelskörpern unter gegenseitigem Einfluss der Gravitation zu programmieren. Als Minimalziel galt es, das Dreikörperproblem zu verstehen, eine animierte Simulation zu programmieren und die Chancen und Grenzen des Modells aufzuzeigen. Als mögliche Erweiterungen standen die Simulation des Dreikörperproblems in einer schnelleren Programmiersprache, beispielsweise Java, sowie die Analyse des Dreikörperproblems mit Körpern unterschiedlicher Masse, mit unterschiedlichen Anfangspositionen und unterschiedlichen Anfangsgeschwindigkeiten, und die Ausweitung der Simulation auf mehr als nur drei Körper (n-Körperproblem) zur Auswahl.

3 Gravitationsprobleme der Himmelsmechanik

3.1 Zweikörperproblem

Das Zweikörperproblem der Himmelsmechanik beschäftigt sich mit der Bewegung zweier Körper, die ohne äusseren Einfluss miteinander interagieren. Sie ziehen sich durch ihre Gravitationskraft gegenseitig an. Durch die Berechnung dieser Anziehungskraft lassen sich die Bahnen der beiden Körper definieren.

3.1.1 Berechnung

Die Berechnung des Zweikörperproblem erfordert ein Integral zweiten Grades nach der Zeit t. Als Ausgangspunkt zur Berechnung dient das folgende zweite Differenzial:

$$\frac{d^2\vec{r}}{dt^2} = \vec{a}(\vec{r}, t)$$

Dabei ist $\vec{r}$ das Wegstück, das der Körper in einer gewissen Zeit t zurücklegt. Der Vektor $\vec{r}$ wird in kartesischen Koordinaten angegeben:

$$\vec{r}(t) = \begin{pmatrix} x(t) \\ y(t) \end{pmatrix}$$

Die Beschleunigung $\vec{a}$, die durch die Gravitationskraft auf einen Körper der Masse m wirkt, lässt sich durch die untenstehende Formel berechnen, wobei G für die Newtonsche Gravitationskonstante steht:

$$\vec{a}(\vec{r}, t) = G \cdot \frac{m}{r^2} \cdot \frac{\vec{r}}{r}$$

Da gemäss Newton III (actio = reactio) beide Körper der Massen m_1 und m_2 die gleiche, einander entgegengesetzte Kraft (Gravitationskraft $\vec{F}_g = G \cdot \frac{m_1 m_2}{r^2} \cdot \frac{\vec{r}}{r}$) erfahren, muss die Beschleunigung für beide Himmelskörper berechnet werden, auch wenn die Masse des einen Himmelskörpers viel grösser ist als die Masse des anderen Himmelskörpers. [1]

3.2 Das Dreikörperproblem

Unter dem Dreikörperproblem wird die Bewegung dreier Himmelskörper unter gegenseitigem Einfluss der Gravitation verstanden. Es ist nicht analytisch lösbar, jedoch kann numerisch eine Näherungslösung berechnet werden.

3.2.1 Vereinfachungen

Für das Verständnis und die Berechnung des Dreikörperproblems werden einige vereinfachende Annahmen getroffen. In dieser Arbeit wird angenommen, dass die Himmelskörper volumenlose Massenpunkte in einem System sind. Da alle hier betrachteten Himmelskörper kugelförmig und symmetrisch sind und (in der Betrachtung unseres Sonnensystems) nicht kollidieren, verfälscht diese Annahme die Berechnungen nicht mehr als die Ungenauigkeit der numerischen Berechnungen. Die Annahme ist ebenfalls nicht mehr zulässig, wenn die Körper nicht starr sind, sondern sich die Körper infolge der Rotation oder Anziehung anderer Himmelskörper verformen [2]. Da die Formen der hier behandelten Himmelskörper nur minimal von einer perfekten Kugel abweichen und sie sich im Zeitraum der Simulation nicht messbar verformen, dürfen die Himmelskörper hier auf Massenpunkte reduziert werden.

3.2.2 Spezialfall des Dreikörperproblems: Gleichseitiges Dreieck

Im allgemeinen Fall verläuft die Bewegung der drei Körper im Dreikörperproblem chaotisch und endet mit einer Kollision. Es gibt jedoch einzelne Spezialfälle, in welchen die Bewegung einer gewissen Regelmässigkeit folgt. In der hier behandelten Arbeit wird nur der Spezialfall des Gleichseitigen Dreiecks erforscht.

1773 erkannte der Physiker Joseph-Louis Lagrange, dass das Dreikörperproblem im Spezialfall von drei Körpern der gleichen Masse, die zu jeder Zeit ein gleichseitiges Dreieck bilden, analytisch gelöst werden kann. [3]

3.2.2.1 Geometrische Darstellung und Eigenschaften

In diesem Fall befinden sich die Körper jeweils an den Ecken des gleichseitigen Dreiecks: Der Körper 1 auf der Ecke A, der Körper 2 auf der Ecke B, und der Körper 3 auf der Ecke C. Die Seitenlänge des gleichseitigen Dreiecks und somit die Distanz zwischen den Körpern wird hier mit s bezeichnet. Die Distanz zwischen den Körpern und dem Massenmittelpunkt M heisst r. Diese Strecke ist gleichzeitig auch der Radius des Umkreises, auf welchem sich die drei Körper bewegen. Die Seitenhalbierende des Dreiecks wird mit a bezeichnet. Der Inkreisradius wird mit b benannt (vgl. Abbildung 1).

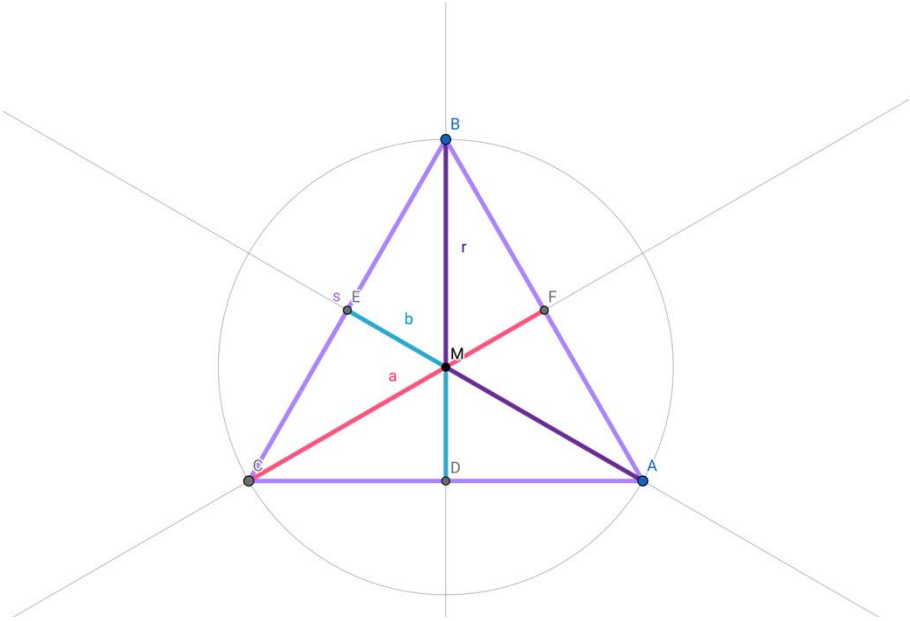


Abbildung 1: Geometrische Darstellung des Spezialfalls des Dreikörperproblem: Gleichseitiges Dreieck

Für die Berechnung der Seitenhalbierenden a in Abhängigkeit der Seitenlänge s wird der Pythagoras verwendet: [4]

$$a = \sqrt{s^2 - \left(\frac{s}{2}\right)^2} = \sqrt{s^2 - \frac{s^2}{4}} = \sqrt{\frac{3}{4}s^2} = \frac{\sqrt{3}}{2}s$$

Da sich die Seitenhalbierenden in einem Verhältnis von $\frac{2}{1}$ schneiden, können die Teilstrecken r und b von a durch a bestimmt werden: [5]

$$r = \frac{2}{3}a = \frac{2}{3} \cdot \frac{\sqrt{3}}{2}s = \frac{\sqrt{3}}{3}s$$

$$b = \frac{1}{3}a = \frac{1}{3} \cdot \frac{\sqrt{3}}{2}s = \frac{\sqrt{3}}{6}s$$

3.2.2.2 Positionen der Körper

Da der Massenmittelpunkt M (Schwerpunkt des gleichseitigen Dreiecks) der Ursprung des Koordinatensystems ausmacht, lautet der Ortsvektor $\overrightarrow{r_1}$ zur Position des Körpers 1 (Punkt A):

$$\overrightarrow{r_1} = \begin{pmatrix} s/2 \\ -b \end{pmatrix}$$

Der Ortsvektor $\overrightarrow{r_2}$ zur Position, bei welcher sich der Körper 2 (Punkt B) befindet, kann wie folgt ausgedrückt werden:

$$\overrightarrow{r_2} = \begin{pmatrix} 0 \\ r \end{pmatrix}$$

Der Ortsvektor $\overrightarrow{r_3}$ zur Position, an welcher sich der Körper 3 (Punkt C) befindet, hat die Komponenten:

$$\vec{r_3} = \begin{pmatrix} -s/2 \\ -b \end{pmatrix}$$

3.2.2.3 Geschwindigkeiten der Körper

Die Geschwindigkeit, welche die drei Körper für die Bewegung um ihren gemeinsamen Schwerpunkt haben müssen, steht stets tangential zum Umkreisradius r. Ihr Betrag kann vom dritten Keplerschen Gesetz abgeleitet werden. Die Winkelgeschwindigkeit, abhängig von den Massen der drei Körper m_1, m_2, m_3 und dem Abstand zwischen den Körpern s, beträgt damit: [6, pp. 7-8]

$$\omega^2 = \frac{G(m_1 + m_2 + m_3)}{s^3}$$

Da die Winkelgeschwindigkeit mit der Geschwindigkeit v und dem Abstand r zwischen dem Körper und dem Massenmittelpunkt ausgedrückt werden kann, kann die obige Gleichung wie folgt umgeformt werden:

$$\frac{v}{r} = \sqrt{\frac{G(m_1 + m_2 + m_3)}{s^3}}$$

Nach dem Auflösen der Gleichung nach der Geschwindigkeit v erhält man folgende Lösung:

$$v = \sqrt{\frac{G(m_1 + m_2 + m_3)}{s^3}}\, r$$

Für die Bestimmung des Vektors der Geschwindigkeit für den Körper 1 auf Punkt A muss die Geschwindigkeit in ihre Komponenten zerlegt werden. (vgl. Abbildung 2) [6, p. 8]

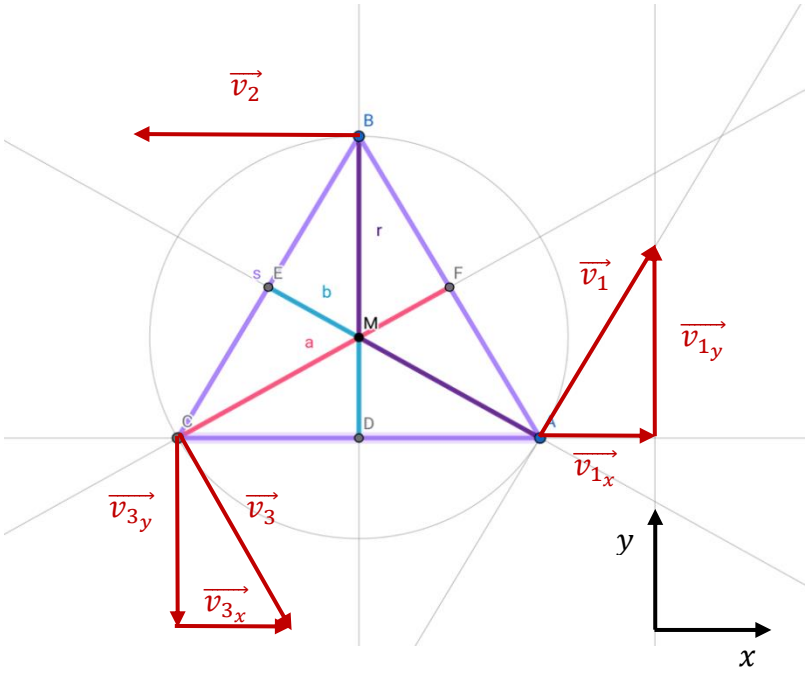


Abbildung 2: Aufteilung der Geschwindigkeitsvektoren im Spezialfall des Dreikörperproblems: Gleichseitiges Dreieck

Der Winkel zwischen dem Vektor $\overrightarrow{v_1}$ und dessen x-Komponente v_{1_x} beträgt aufgrund der Symmetrie des gleichseitigen Dreiecks 60°. Daher betragen v_{1_x} und v_{1_y}:

$$\left\|\overrightarrow{v_{1_x}}\right\| = \cos(60°) \cdot v$$

$$\left\|\overrightarrow{v_{1_y}}\right\| = \sin(60°) \cdot v$$

Der Vektor $\overrightarrow{v_1}$ lautet somit $\overrightarrow{v_1} = \begin{pmatrix} v_{1_x} \\ v_{1_y} \end{pmatrix}$ oder $\overrightarrow{v_1} = \begin{pmatrix} \cos(60°) \cdot v \\ \sin(60°) \cdot v \end{pmatrix}$.

Da der Geschwindigkeitsvektor $\overrightarrow{v_2}$ des Körpers 2 auf Punkt B antiparallel zur x-Achse verläuft, beträgt dessen y-Komponente $v_{2_y} = 0$ und deren x-Komponente $v_{2_x} = -v$.

Der Vektor $\overrightarrow{v_2}$ kann daher wie folgt ausgedrückt werden: $\overrightarrow{v_2} = \begin{pmatrix} v_{2_x} \\ v_{2_y} \end{pmatrix}$ oder $\overrightarrow{v_2} = \begin{pmatrix} -v \\ 0 \end{pmatrix}$.

Der Winkel zwischen dem Vektor $\overrightarrow{v_3}$, der Geschwindigkeit des Körpers 3 auf Punkt C, und dessen x-Komponente v_{3_x} beträgt 60°. Die x-Komponente v_{3_x} verläuft parallel zur x-Achse und die y-Komponente v_{3_y} antiparallel zur x-Achse. Daher betragen die Komponenten des Vektors $\overrightarrow{v_3}$:

$$\left\|\overrightarrow{v_{3_x}}\right\| = \cos(60°) \cdot v$$

$$\left\|\overrightarrow{v_{3_y}}\right\| = -\sin(60°) \cdot v$$

Der Vektor $\overrightarrow{v_3}$ lautet also: $\overrightarrow{v_3} = \begin{pmatrix} v_{3_x} \\ v_{3_y} \end{pmatrix}$ oder $\overrightarrow{v_3} = \begin{pmatrix} \cos(60°) \cdot v \\ -\sin(60°) \cdot v \end{pmatrix}$.

3.2.2.3.1 Umlaufzeit

Die Umlaufzeit T kann ausgedrückt werden durch:

$$T = \frac{2\pi}{\omega}$$

In 3.2.2.3 wurde mit dem Keplerschen Gesetz die Winkelgeschwindigkeit ω wie folgt bestimmt:

$$\omega^2 = \frac{G(m_1 + m_2 + m_3)}{s^3}$$

Damit lässt sich die Umlaufzeit T in diesem Spezialfall des Dreikörperproblems mit der untenstehenden Formel berechnen:

$$T = 2\pi \sqrt{\frac{s^3}{G(m_1 + m_2 + m_3)}}$$

3.3 Das n-Körperproblem

3.3.1 Berechnungen

3.3.1.1 Gravitationskraft im n-Körperproblem

Für die Berechnung der Gravitationskraft auf den Körper *i* werden die Gravitationskräfte zwischen Körper *i* und den anderen *n-1* Körpern gemäss dem Newtonschen Gravitationsgesetz aufsummiert: [7], [8]

$$\vec{F_i} = -G \sum_{\substack{j=1 \\ j \neq i}}^{n} m_i m_j \frac{\vec{r_i} - \vec{r_j}}{\left\|\vec{r_i} - \vec{r_j}\right\|^3}$$

3.3.1.2 Gravitationsbeschleunigung im n-Körperproblem

Aus der obigen Formel für die Kraft, die auf den Körper *i* wirkt, kann die Formel für die Gravitationsbeschleunigung abgeleitet werden. Dazu wird Newtons zweites Axiom $\vec{F_i} = m_i \vec{a_i}$ hinzugezogen. Die Beschleunigung $\vec{a_i}$, welche durch die Summe der Gravitationskräfte auf den Körper *i* wirkt, kann durch die Division der Masse m_i aus der Gleichung für die resultierende Kraft auf den Körper *i* im n-Körperproblem hergeleitet werden (vgl. 3.3.1.1). [7]

$$\vec{a_i} = -G \sum_{\substack{j=1 \\ j \neq i}}^{n} m_j \frac{\vec{r_i} - \vec{r_j}}{\left\|\vec{r_i} - \vec{r_j}\right\|^3}$$

3.3.1.3 Potentielle Energie im n-Körperproblem

Die potentielle Energie im n-Körperproblem ist die Summe der Potentiale zwischen allen n Körper. Das Potential zwischen den zwei Körpern *1* und *2* lautet: [9]

$$U = -G \frac{m_1 m_2}{\vec{r}}$$

Das gesamte Potential im n-Körperproblem lautet folglich: [7], [8, p. 18]

$$U = -\frac{1}{2} G \sum_{i=1}^{n} \sum_{j \neq i} \frac{m_i m_j}{\left\|\vec{r_i} - \vec{r_j}\right\|}$$

3.3.1.4 Kinetische Energie im n-Körperproblem

Die kinetische Energie im n-Körperproblem ist die Summe der kinetischen Energien aller Körper. Die kinetische Energie eines Körpers lautet:

$$\overrightarrow{E_{kin}} = \frac{1}{2} m \vec{v}^2$$

Daher kann die gesamte kinetische Energie im n-Körperproblem wie folgt berechnet werden: [7], [8, p. 23]

$$I = \sum_{i=1}^{n} \frac{1}{2} m_i \|\vec{v_i}\|^2$$

3.3.1.5 Impuls im n-Körperproblem

Der gesamte Impuls im System ist die Summe der Impulse aller Körper. Dieser wird definiert durch:

$$\vec{p} = m \cdot \vec{v}$$

Damit kann die Summe aller Impulse wie folgt beschrieben werden:

$$\overrightarrow{p_{ges}} = \sum_{i=1}^{n} m_i \overrightarrow{v_i}$$

3.4 Das Newtonsche Gravitationsgesetz

Ende des 17. Jahrhunderts erkannte Isaac Newton, dass Massen miteinander wechselwirken. Diese Wechselwirkung wird umso grösser, je grösser die miteinander wechselwirkenden Massen sind. Die Kraft, die durch diese Wechselwirkung auf die beiden Massen wirkt, nennt man Gravitation.

Newton beobachtete, dass die Gravitationskraft linear grösser wird, je grösser die Massen sind. Ausserdem stellte er fest, dass die Gravitation mit dem Abstandsquadrat abnimmt. Daraus leitete er folgende Formel her: [10], [11]

$$\vec{F} = G \frac{m \cdot M}{\vec{r}^2}$$

Dabei ist G eine noch unbekannte Konstante mit der SI-Einheit $[G] = \frac{m^3}{kg \cdot s^2}$. Der Zahlenwert von G wurde etwa 100 Jahre nach Newton erstmals von Henry Cavendish gemessen. Sie beträgt gemäss modernen Messungen etwa $6.67 \cdot 10^{-11} \frac{m^3}{kg \cdot s^2}$. [12]

3.4.1 Grenzen des Newtonschen Gravitationsgesetzes

Das Newtonsche Gravitationsgesetz unterliegt der Allgemeinen Relativitätstheorie von Einstein. Deshalb darf das Gravitationsgesetz nur dann angewandt werden, wenn der Effekt der Relativitätstheorie (fast) nichts am System ändert. Auf das Dreikörperproblem bezogen heisst das, dass die Geschwindigkeit der Himmelskörper nicht schneller als einen Zehntel der Lichtgeschwindigkeit c sein darf und die Massen der Himmelskörper nicht allzu kompakt sein dürfen, damit sie den umliegenden Raum und die Zeit nicht zu stark krümmen. [10]

4 Programmiermethoden

4.1 Der Leapfrog-Algorithmus

Für die Berechnung der Bewegung von Himmelskörpern (Dreikörperproblem und n-Körperproblem) muss numerisch integriert werden. Dafür wird der Integrations-Algorithmus

Leapfrog verwendet. Das Leapfrog-Verfahren besteht aus einem Zeitschritt Δt. Während einem Zeitschritt Δt wird die Position eines Körpers zweimal berechnet. Für das erste Positionsupdate (Strecke zwischen Startposition $\vec{r}_{start}$ und Zwischenposition $\vec{r}_{zwischen}$) wird die Geschwindigkeit $\vec{v}_{start}$ des Körpers zum Zeitpunkt t berechnet. Die Zwischenposition $\vec{r}_{zwischen}$ wird dann mit dieser Geschwindigkeit $\vec{v}_1$ über einen halben Zeitschritt $\Delta t/2$ berechnet:

$$\vec{r}_{zwischen} = \vec{r}_{start} + \vec{v}_{start} \cdot \Delta t/2$$

Mit den Zwischenpositionen der beiden Körper wird die neue Beschleunigung berechnet. Durch diese lässt sich die neue Geschwindigkeit $\vec{v}_{neu}$ über einen ganzen Zeitschritt Δt hinweg berechnen:

$$\vec{v}_{neu} = \vec{v}_{start} + \vec{a} \cdot \Delta t$$

Mit dieser neuen Geschwindigkeit $\vec{v}_2$ wird die Position ein zweites Mal aktualisiert und so die Endposition $\vec{r}_{end}$ berechnet:

$$\vec{r}_{end} = \vec{r}_{zwischen} + \vec{v}_{neu} \cdot \Delta t/2$$

Durch den Leapfrog-Algorithmus lassen sich die Positionen der Himmelskörper mit verhältnismässig wenig Rechenschritten relativ genau bestimmen. [13]

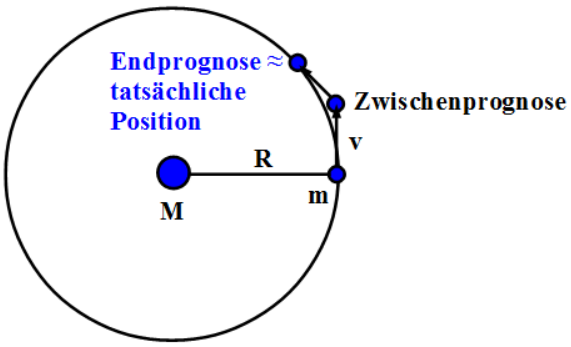


Abbildung 3: Startposition, Zwischenposition, Endposition eines Zeitschritts gemäss Leapfrog [13]

Die Rechenschritte sind in der obigen Grafik (vgl. Abbildung 3) dargestellt. Die Zwischenposition wird unter der Annahme berechnet, dass der Körper sich in einer gleichförmig geradlinigen Bewegung (mit konstanter Geschwindigkeit) bewegt. Der Geschwindigkeitsvektor $\vec{v}$ steht senkrecht auf dem Abstandsvektor $\vec{R}$. Die aus der Berechnung folgende Zwischenposition (Zwischenprognose) liegt deutlich abseits des Orbits. Von da aus wird die neue Gravitationskraft mit dem neuen Abstandsvektor und die daraus folgende neue Beschleunigung berechnet. Die Beschleunigung verändert die Richtung der Geschwindigkeit $\vec{v}$. Mit der neuen Geschwindigkeit, die wiederum als konstant betrachtet wird, wird die Endposition berechnet, die, anders als die Zwischenposition, ziemlich exakt auf der Umlaufbahn liegt und deshalb ungefähr der tatsächlichen Position entspricht. [13]

4.1.1 Energieerhaltung des Leapfrog-Algorithmus

Im Leapfrog-Algorithmus wird die Energie nicht perfekt erhalten. Stattdessen oszilliert sie, stärker, je grösser der Zeitschritt Δt gewählt wird. Dabei bewirkt eine Halbierung des Zeitschritts, dass die Schwankungen in der Gesamtenergie geviertelt werden. Trotzdem wird die Energie mit dem Leapfrog-Algorithmus erhalten, da durch die Oszillation die Schwankungen sich in einem limitierten Rahmen bewegen und die Gesamtenergie nicht stetig anwächst oder abfällt. [14]

4.2 Der Python-Dictionary

In Python wird ein Dictionary benutzt, um Daten sortiert zu speichern. Dazu werden die Daten unter einem Key gespeichert. Dieser muss zwingend ein String sein. Der Datentyp des Eintrags unter dem für den Dictionary spezifischen String ist jedoch beliebig, es können Listen, Tuples, Arrays, Integers, Floats, Strings oder sogar ein weiterer Dictionary in einem Dictionary gespeichert werden.

4.2.1 Erstellen eines Dictionary‘s

Initiiert wird ein Dictionary mit geschweiften Klammern:

```
example = {}
```

Der Name des Dictionary‘s lautet *example*. Noch ist dieser leer. Im Folgenden wird ein Eintrag (eine Liste) unter dem Key *‘start_values‘* mit verschiedenen Integers in den Dictionary eingefügt.

```
example ['start_values'] = [1,2,3,4,5]
```

In einem Dictionary können beliebig viele Einträge unter beliebig vielen Keys (gleich viele Einträge wie Keys) gespeichert werden. Die Einträge können beliebige Datentypen sein, es kann sogar in einem Dictionary ein weiterer Dictionary initiiert werden. Der neue Eintrag, ein Dictionary wird unter dem Key *‘new_dict‘* eingefügt.

```
example ['new_dict'] = {}
```

In diesen können nun auch wieder beliebig viele Einträge unter beliebig vielen Keys eingefügt werden.

```
example ['new_dict']['goal'] = True
```

4.2.2 Zugriff auf Daten in einem Dictionary

Zugriff auf die gespeicherten Einträge wird erlangt, indem der Name des Dictionary‘s gefolgt vom Key (String in eckigen Klammern) eingegeben wird. In diesem Beispiel wird der Eintrag in der sogenannten *Shell*, dem Ausgabefenster des Compilers ausgegeben.

```
print(example['start_values'])
```

Aufgrund des obigen Einfügens der Liste unter dem Key *'start_values'*, wird in der Shell folgende Liste erscheinen:

```
[1,2,3,4,5]
```

Um auf ein bestimmtes Element aus dieser Liste zuzugreifen, muss der Index des spezifischen Elements hinten angehängt werden. Dieser gibt die Position des Elements in einer Liste an. Das erste Element einer Liste besitzt den Index 0:

```
print(example['start_values'][2])
```

In der Shell wird folgende Zahl (Integer) erscheinen:

```
3
```

Um Zugriff auf Einträge im Dictionary *new_dict*, welcher im Dictionary *example* initiiert wurde, zu erlangen, muss der Key (String in eckigen Klammern) des äusseren Dictionary's *example* vor den Key des inneren Dictionary's *' new_dict'* gestellt werden:

```
print(example['new_dict']['goal'])
```

Folgender Boolean Value wird in der Shell erscheinen:

```
True
```

4.2.3 Bearbeitung eines Dictionary-Eintrags

Ein Eintrag in einem Dictionary kann beliebig modifiziert werden. Der Liste, die unter dem Key *'start_values'* im Dictionary *example* gespeichert ist, kann beispielsweise ein weiterer Wert hinzugefügt werden:

```
example['start_values'].append(6)
```

Wenn nun der Eintrag unter dem Key *'start_values'* im Dictionary *example* in der Shell ausgegeben wird, erscheint folgende Liste: [15]

```
[1,2,3,4,5,6]
```

4.3 Matplotlib

Für die graphische Darstellung von Daten wird die Bibliothek *matplotlib.pyplot* verwendet. Diese ermöglicht eine zwei- oder dreidimensionale Darstellung der Daten.

4.3.1 Daten plotten

Die Bibliothek wird importiert mit:

```
import matplotlib.pyplot as plt
```

Um Daten plotten zu können, muss zuerst eine graphische Oberfläche erstellt werden. Dies geschieht mit dem folgenden Code-Ausschnitt: [16]

```
fig = plt.figure()
```

Es können beliebig viele Subplots erstellt werden. Im Folgenden wird nur ein Subplot erstellt. Die ersten beiden 1 bedeuten, dass es $1 \cdot 1$ Subplots gibt. Das dritte 1 bedeutet, dass mit dem Attribut *ax* der erste (und einzige) Subplot erstellt wird. Mit *projection = '3d'* wird der Plot dreidimensional gemacht. Wird das Argument *projection* ausgelassen, ist der Plot zweidimensional: [17]

```
ax = fig.add_subplot(1,1,1, projection = '3d')
```

Die eigentlichen Daten werden mit *ax.plot* dargestellt. Die Funktion verlangt 3 Argumente: eine Liste mit den x-Koordinaten, eine Liste mit den y-Koordinaten und eine Liste mit den z-Koordinaten. Wenn nur ein einzelner Punkt dargestellt werden soll, kann auch nur eine einzelne x-, y- oder z-Koordinate angegeben werden. Die Funktion kann mehr als einmal aufgerufen werden, um mehrere Datensätze im gleichen Plot darzustellen. Wenn unter den Variablen *X*, *Y* und *Z* Listen mit den jeweiligen Koordinaten gespeichert sind, kann der Datensatz folgendermassen dargestellt werden. Wichtig dabei ist, dass die Listen *X*, *Y*, und *Z* alle gleich viele Elemente beinhalten und dass die Elemente Integers oder Floats sind:

```
ax.plot(X, Y, Z)
```

Als zusätzliche Argumente können die Farbe (mit *color* =), die Linienart (mit *linestyle* =), den Marker für einen Punkt (mit *marker* =), die Grösse des Markers (mit *markersize* =, oder *ms* =) oder die Bezeichnung des Plots für die Legende (mit *label* =) angegeben werden. [18]

Die bereits existierenden Farben und deren Namen können an der folgenden von Matplotlib stammenden Tabelle abgelesen werden (vgl. Abbildung 4):

Abbildung 4: Farben in Matplotlib [19]

Mit der folgenden Linie kann die Legende dargestellt werden. Die Farbe und die Bezeichnung für den bezeichneten Plot wird aus den Argumenten für *ax.plot* entnommen: [20]

```
ax.legend()
```

Dem Plot kann ein Titel hinzugefügt werden mit: [21]

```
ax.set_title ('Dreikörperproblem')
```

Mit den folgenden Linien Code können die x-, y- und z-Achse beschriftet werden: [22]

```
ax.set_xlabel ('x [AE]')
ax.set_ylabel ('y [AE]')
ax.set_zlabel ('z [AE]')
```

Der folgende Code-Ausschnitt dient der Ausgabe des Plots: [23]

```
plt.show()
```

4.3.2 Animierter Plot

Um einen Plot zu animieren, werden in einer *for*-Schleife Teile des Codes graphisch dargestellt. Es können nur Punkte, aber auch Linien ausgegeben werden. Die eigentliche Darstellung der Daten erfolgt wie im Abschnitt 4.3.1 beschrieben. Der einzige Unterschied besteht darin, dass an Stelle von *plt.show* die Funktion *plt.pause* verwendet wird. Das

Argument dieser Funktion ist die Zeit, für welche der Plot (= Frame der Animation) angezeigt werden soll, in Sekunden. Mit dem folgenden Code-Ausschnitt wird jeder Frame für 0.001 Sekunden angezeigt, bevor der nächste darüber gezeichnet wird: [24]

```
plt.pause(0.001)
```

Normalerweise bleibt das, was in einem Frame dargestellt ist, für den Rest der Simulation sichtbar und die nächsten Frames werden einfach darübergelegt. Wenn aber der vorherige Frame nicht mehr zu sehen sein soll, sobald der nächste Frame erscheint, kann mit der Funktion *plt.cla* der Frame geschlossen werden. Mit dieser Ergänzung ist immer nur der aktuelle Frame sichtbar und nicht auch alle vorhergegangenen: [25]

```
plt.cla()
```

5 Aufbau der Simulation

Für die Simulation wird der Programmcode von fünf verschiedenen Files verwendet. Diese werden in den Abschnitten 5.1, 5.2, 5.3, 5.4 und 5.5 abschnittsweise beschrieben und erklärt. Alle Files befinden sich auf dem USB-Stick MIKUELLING im Ordner *Code*. Der zusammenhängende Code der Files ist im Anhang einzusehen (vgl. 10.3).

5.1 Simulation des n-Körperproblems

Die Berechnungen für das n-Körperproblem sowie die Darstellung der Resultate erfolgen im File *n_Körperproblem_Berechnungen*. In den folgenden Unterabschnitten wird der Code in Abschnitten beschrieben und erklärt.

5.1.1 Importe

Zu Beginn des Programms wird die Bibliothek *numpy* importiert. Sie beinhaltet die Arrays, welche für die Vektoren der Geschwindigkeiten, Positionen, Beschleunigungen, usw. verwendet werden.

Die Bibliothek *matplotlib.pyplot* wird für die Darstellung der Ergebnisse, sowohl für die Bewegungen der Körper wie auch für die Entwicklung der potentiellen und kinetischen Energie im System und der Energie- und Impulserhaltung verwendet.

Von *scipy.constants* werden die Grössen *au*, welche die Anzahl Meter pro Astronomische Einheit beinhaltet und *G*, welche den Zahlenwert der Newtonschen Gravitationskonstante in SI-Einheiten enthält, importiert. Die Grösse *au* wird für die Umwandlung der Koordinaten der Positionen von m in AE verwendet. Die Newtonsche Gravitationskonstante wird für die Berechnung der Beschleunigung durch die Gravitation und für die Berechnung der potentiellen Energie im System verwendet.

Das File *daten_speichern* enthält die Funktionen *save_csv_array*, welche verwendet wird, um die Geschwindigkeiten und Positionen aller Körper in einem csv-File (comma separated values) als Arrays zu speichern, und *save_csv_list*, welche eine Liste von Floats wie die potentiellen und kinetischen Energien, die Gesamtenergien und die Gesamtimpulse über den Zeitraum der Berechnung in einem csv-File speichert. (vgl. 5.4)

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.constants import au, G
import daten_speichern as ds
```

5.1.2 Funktionen

Die ganze Simulation ist in der Funktion *n_Körperproblem* verpackt. Sie erfordert sieben Parameter.

Der erste Parameter ist der Dictionary *daten* mit allen notwendigen Grössen der Himmelskörper (vgl. 5.5). Der zweite Parameter ist *n* und beschreibt die Anzahl verwendeter Körper als Integer. Der Parameter *Gesamtlaufdauer* ist ein Integer oder Float und gibt die Zeit in Sekunden an, über welche die Berechnungen durchgeführt werden sollen. Der Parameter *delta_T* ist ein Integer und bestimmt den Zeitschritt für die Berechnung mit dem Leapfrog-Algorithmus (vgl. 4.1). Der Parameter *Fixierung* ist ein Boolean. Wenn dieser *True* ist, wird das Programm so fixiert, dass der Körper mit dem Key *'0'* im Dictionary Daten sich stets am Ursprung des Koordinatensystems befindet. Der Parameter *Vergleich* ist ebenfalls ein Boolean Value. Wenn er *True* ist, werden die aus der Theorie erwarteten Endpositionen zusätzlich zu den berechneten Endpositionen in grün auf dem Plot dargestellt. Der Parameter *Zusatz* dient der Benennung der csv-Files und wird als zusätzliches Kürzel hinter den Namen des Files angehängt. Im Folgenden ist die Dokumentation der Funktion *n_Körperproblem* als Docstring aufgeführt.

```
def n_Körperproblem(daten, n, Gesamtlaufdauer, delta_T, Fixierung,
Vergleich, Zusatz):
    '''
    Input Parameter:
    daten:
        Dictionary mit Daten aller n Himmelskörper
        Für jeden Himmelskörper eigener Dictionary in daten mit Key '0',
'1', ... str(n-1) mit:
            Masse (Key 'm')
            Anfangsposition (Key 'r')
            Anfangsgeschwindigkeit (Key 'v')
            Farbe des Körpers (Key 'farbe')
            Relative Grösse des Körpers (Key 'grösse')
            Name des Körpers (Key 'name')
            aus der Theorie erwartete Endposition nach Laufdauer des
Programms (Key 'end_r') wenn Vergleich == True
    n:
        Anzahl verwendeter Körper

    Gesamtlaufdauer:
        Gesamtzeit, über welche das Programm die Berechnungen durchführen
soll in Sekunden

    delta_T:
        Zeitschritt für die Berechnung mit Leapfrog-Algorithmus in Sekunden

    Fixierung:
```

```
        Wenn True, wird die Position des Körpers mit Eintrag in daten['0']
am Ursprung fixiert

    Vergleich:
        Wenn True, werden im Plot die aus der Theorie erwarteten
Endpositionen zusätzlich
        zu den berechneten Endpositionen geplottet

    Zusatz:
        Kürzel am Ende der Namen für die csv-Files als String

    Returns:
    csv-Files mit
        berechneten Positionen und Geschwindigkeiten der Körper,
        berechneten Werten der kinetischen und potentiellen Energie
        berechneten Werten der Gesamtenergie und der Gesamtimpulses

    Plot mit berechnetem n-Körperproblem, Plot zur Energieerhaltung, Plot
zur Impulserhaltung
    '''
```

5.1.2.1 Beschleunigung durch Gravitation

In der Funktion *n_Körperproblem* ist die Funktion *a_i* für die Berechnung der Gravitationsbeschleunigung definiert. Diese verlangt den Parameter *i*, einen Integer, welcher den Körper bestimmt, für welchen die Beschleunigung berechnet werden soll. Die Gravitationsbeschleunigung wird mit der Formel aus 3.3.1.2 berechnet:

$$\vec{a_i} = -G \sum_{\substack{j=0 \\ j \neq i}}^{n-1} m_j \frac{\vec{r_i} - \vec{r_j}}{\left\|\vec{r_i} - \vec{r_j}\right\|^3}$$

Um alle Beschleunigungen aufzusummieren, wird eine *for*-Schleife eingefügt. Da die Körper mit den Zahlen von *0* bis *n-1* benannt worden sind, startet *j* bei 0. Die Bedingung, dass $j \neq i$, wird durch ein *if*-Statement erfüllt. Die Masse des Körpers *j* wird aus dem Dictionary des Körpers *j* entnommen. Da im Leapfrog-Algorithmus die neue Beschleunigung im Zwischenschritt berechnet wird, wird für $\vec{r_i}$ und $\vec{r_j}$ die berechnete Zwischenposition verwendet, welche im jeweiligen Dictionary unter dem Key *'r_zwischen'* gespeichert ist. Die Berechnung des Terms im Summenzeichen erfolgt in der *for*-Schleife so, wie er in der Formel angegeben wird. Um die korrekte Rechenabfolge zu gewährleisten, werden Klammern verwendet. Für die Berechnung des Betrags eines Vektors wird die Funktion *np.linalg.norm*, welche aus der Bibliothek *numpy* importiert wurde, verwendet.

Nach der Summierung aller Terme, wird das Ergebnis gemäss der Formel mit -*G* multipliziert. Dieser Wert wird darauf mit *return* zurückgegeben. Die Funktion *a_i* sieht in Python wie folgt aus:

```
def a_i(i):
    '''
    Input Parameter:
```

```
    i: Nummer des Körpers, für welchen die Beschleunigung berechnet werden
soll

    Returns:
    berechnete Beschleunigung des Körpers i'''
    a_sum = 0
    for j in range(n):
        if i == j:
            continue
        else:
            a_sum += daten[str(j)]['m'] * ((daten[str(i)]['r_zwischen'] -
daten[str(j)]['r_zwischen'])/ (np.linalg.norm(daten[str(i)]['r_zwischen'] -
daten[str(j)]['r_zwischen']))**3)
    a_sum *= -G
    return a_sum
```

5.1.2.2 Potentielle Energie im System

Mit der Funktion *potentielle_Energie* wird die summierte potentielle Energie aller Körper berechnet. Diese Funktion erfordert keine Parameter. Die Berechnung erfolgt dabei mit der folgenden Formel aus 3.3.1.3:

$$U = -\frac{1}{2}G\sum_{i=0}^{n-1}\sum_{j\neq i}\frac{m_i m_j}{\|\vec{r_i}-\vec{r_j}\|}$$

Für die beiden Summenzeichen der Formel werden zwei *for*-Schleifen eingesetzt. Da die Nummerierung der Körper bei 0 anfängt, startet der Wert für den Parameter *i* der äusseren *for*-Schleife (= erstes Summenzeichen) bei *0* und geht bis *n-1*. In der inneren *for*-Schleife (= zweites Summenzeichen) werden für den Parameter *j* ebenfalls Integers von *0* bis *n-1* eingesetzt, wobei ein *if*-Statement sicherstellt, dass *i* und *j* nicht den gleichen Wert annehmen. Die Berechnung von $\|\vec{r_i}-\vec{r_j}\|$ wird hier, wie schon in 5.1.2.1 mit der Funktion *np.linalg.norm* vorgenommen. In der inneren *for*-Schleife wird der Term $\frac{m_i m_j}{\|\vec{r_i}-\vec{r_j}\|}$ berechnet und der Summe angefügt. Die Massen m_i und m_j stammen aus dem jeweiligen Dictionary. Nach den beiden *for*-Schleifen wird die Summe der Terme mit $-\frac{1}{2}G$ multipliziert und mit *return* zurückgegeben. Der Python-Code für die Funktion *potentielle_Energie* sieht wie folgt aus:

```
def potentielle_Energie():
    '''
    Input Parameter:
    None

    Returns:
    potentielle Energie im System'''

    pot_sum = 0
    for i in range(n):
        for j in range(n):
            if i == j:
                continue
            else:
```

```
                pot_sum += (daten[str(i)]['m'] * daten[str(j)]['m'])/
(np.linalg.norm(daten[str(i)]['r_end'] - daten[str(j)]['r_end']))
    pot_sum *= -(1/2)*G
    return pot_sum
```

5.1.2.3 Kinetische Energie im System

Für die Berechnung der kinetischen Energie im System wird nach der in 3.3.1.4 eingeführten Formel berechnet:

$$I = \sum_{i=0}^{n-1} \frac{1}{2} m_i \|\vec{v_i}\|^2$$

Da die Nummerierung der Körper bei 0 startet, startet auch der Wert von *i* bei 0 und geht dafür nur zu *n-1*. In einer *for*-Schleife, in welcher der Parameter *i* die Werte *0* bis *n-1* durchläuft, wird für jeden einzelnen Körper die kinetische Energie gemäss der obigen Formel berechnet. Der die Berechnung der Norm $\|\vec{v_i}\|$ wird dabei mit der Funktion *np.linalg.norm* durchgeführt. Nach der Aufsummierung wird das Resultat mit *return* zurückgegeben. Hier ist der Code für die Funktion *kinetische_Energie* aufgeführt:

```
def kinetische_Energie():
    '''Input Parameter:
    None

    Returns:
    kinetische Energie im System'''

    kin_sum = 0
    for i in range(n):
        kin_sum += (1/2) * daten[str(i)]['m'] *
(np.linalg.norm(daten[str(i)]['v_new']))**2

    return kin_sum
```

5.1.2.4 Gesamtenergie im System

Die Funktion Gesamtenergie addiert die kinetische und die potentielle Energie im System. Dafür ruft diese die beiden Funktionen auf und gibt die Summe der beiden Energien mit *return* zurück.

```
def Gesamtenergie ():
    '''
    Input Parameter:
    None

    Returns:
    Gesamtenergie im System'''
    return potentielle_Energie() + kinetische_Energie()
```

5.1.2.5 Gesamtimpuls im System

Die Funktion *Gesamtimpuls* erfordert keine Parameter. Sie beherbergt die Funktion *Impuls*, welche als Parameter die Masse eines Körpers in kg und seine Geschwindigkeit als

dreidimensionalen Array in kartesischen Koordinaten verlangt. Er berechnet den Impuls des Körpers mit $\overrightarrow{p_i} = m \cdot \overrightarrow{v_i}$ und gibt den berechneten Wert zurück.

Die Funktion *Gesamtimpuls* ruft mit einer *for*-Schleife die Funktion *Impuls* für jeden Körper auf und summiert die Impulse der Körper auf. Damit der Impuls der Körper sich nicht gegenseitig aufhebt, wird für das Dreikörperproblem die Norm des Impulses aufsummiert, während im allgemeinen n-Körperproblem die Vektoren aufsummiert und die Norm davon erst am Schluss berechnet wird. Zuletzt wird die Norm des Gesamtimpulses mit *return* zurückgegeben. Unten ist der Python-Code für die Funktion *Gesamtimpuls* aufgeführt:

```
def Gesamtimpuls ():
    '''
    Input Parameter:
    None

    Returns:
    Summe der Impulse der Körper 1, 2, 3 in N*s'''

    def Impuls (m,v):
        '''
        Input Parameter:
        m: Masse eines Körpers in kg
        v: dreidimensionaler Array desselben Körpers als kartesische
Koordinaten in m/s

       Returns:
       Betrag des berechneten Impulses des Körpers in N*s'''

       return m * v

    p_ges = 0
    if n == 3:
        for j2 in range (n):
            p_ges += np.linalg.norm(Impuls(daten[str(j2)]['m'],
daten[str(j2)]['v_new']))
    else:
        for j3 in range (n):
            p_ges +=Impuls(daten[str(j3)]['m'], daten[str(j3)]['v_new'])

    return np.linalg.norm(p_ges)
```

5.1.2.6 Berechnung mit dem Leapfrog-Algorithmus

Die Funktion *leapfrog* verlangt zwei Parameter. Der Parameter *T* definiert die gesamte Zeitspanne für die Berechnung in *s*. Der Parameter *dt* bestimmt die Grösse des Zeitschritts Δt in *s* für die Berechnung mit dem numerischen Integrationsalgorithmus *Leapfrog*. Unten ist die Funktion *leapfrog* als Docstring aufgeführt. (vgl. 4.1)

```
def leapfrog(T, dt):
    '''
    Input Parameter:
    T: Zeitspanne für die Berechnung in s (Float oder Integer)
    dt: Grösse der Zeitschritte in s (Integer)
```

```
    Returns:
    Tuple mit:
      Liste mit den kinetischen Energien über die Zeitdauer der Berechnung
      Liste mit den potentiellen Energien über die Zeitdauer der Berechnung
      Liste mit den Gesamtenergien über die Zeitdauer der Berechnung
      Liste mit den Gesamtimpulsen über die Zeitdauer der Berechnung

    Einträge der Positionen und Geschwindigkeiten in den Dictionary der
Körper '''
```

In der Funktion *leapfrog* wird zunächst im Dictionary von jedem Körper innerhalb des Dictionary's *daten* eine Liste unter dem Key *'R'* eingefügt. Das erste Element dieser Liste ist die Startposition des jeweiligen Körpers. Alle weiteren berechneten Positionen werden im Verlauf der Berechnung in diese Liste eingefügt werden.

```
# unter dem Key 'R' findet sich eine Liste von Positionen der Körper 0, 1,
..., n
# Hier wird die Startposition der Körper 0, 1, ..., n eingefügt
for i1 in range(n):
    daten[str(i1)]['R'] = [daten [str(i1)]['r'] ]
```

Analog zur Liste für die Positionen wird auch im Dictionary von allen Körpern eine Liste für die Geschwindigkeiten über die Zeitdauer der Berechnung eingefügt, welche bereits den Array der Startgeschwindigkeit als ersten Parameter beinhaltet.

```
# unter dem Key 'V' findet sich eine Liste von Geschwindigkeiten (als
dreidimensionaler Array) der Körper 0, 1, ..., n
# Hier wird die Startgeschwindigkeit der Körper 0, 1, ..., n eingefügt
for i2 in range(n):
    daten[str(i2)]['V'] = [daten [str(i2)]['v'] ]
```

Für die Energien im System wird jeweils eine Liste erstellt, in welche die berechneten Energien eingefügt werden sollen. Ausserdem wird eine Liste für die Speicherung aller Gesamtimpulse über die Zeitdauer der Berechnung eingefügt.

```
# Energieberechnung
# Liste für potentielle Energie im System
E_pot = []
# Liste für kinetische Energie im System
E_kin = []
# Liste für Gesamtenergie
E_G = []

# Impulsberechnung
# Liste für Gesamtimpuls
p_G = []
```

In der folgenden *for*-Schleife wird die eigentliche Berechnung mit dem Leapfrog-Algorithmus durchgeführt. (vgl. 4.1)

```
for j in range (0, int(T), dt):
```

```
    # Für alle Körper 0, 1, ..., n werden die Zwischenpositionen gemäss
Leapfrog (r2 = r1 + v1 * dt/2)
    for i3 in range(n):
         daten[str(i3)]['r_zwischen'] = daten[str(i3)]['R'][-1] +
daten[str(i3)]['V'][-1] * dt/2

    # Für alle Körper 0, 1, ..., n werden die Beschleunigungen durch die
Gravitationskräfte berechnet (Funktion: a_i)
    for i4 in range(n):
        daten[str(i4)]['a'] = a_i(i4)

    # Für alle Körper 0, 1, ..., n werden die neuen Geschwindigkeiten
berechnet gemäss Leapfrog (v2 = v1 + a * dt)
    for i5 in range(n):
        daten[str(i5)]['v_new'] = daten[str(i5)]['V'][-1] +
daten[str(i5)]['a'] * dt

    # Für alle Körper 0, 1, ..., n werden die Endpositionen berechnet
gemäss Leapfrog (r3 = r1 + v2 * dt/2)
    for i6 in range(n):
        daten[str(i6)]['r_end'] = daten[str(i6)]['r_zwischen'] +
daten[str(i6)]['v_new'] * dt/2
```

Falls der Input Parameter *Fixierung* der Funktion *n_Körperproblem* auf *True* gesetzt wurde, wird im Folgenden die Simulation an der Position des Körpers 0 fixiert. Es werden alle Körper so verschoben, dass der Körper mit dem Key *'0'* sich im Ursprung des Koordinatensystems befindet. Die Geschwindigkeiten werden dabei nicht verändert, um die Energie- und Impulserhaltung zu gewährleisten.

```
# Falls Fixierung == True, Fixierung der Simulation an der Position des
Körpers '0'
# -> Verschiebung aller Koordinaten, sodass sich die Körper '0' sich bei r
= [0,0,0] befindet
if Fixierung:
    delta_r = np.array([0,0,0]) - daten['0']['r_end']
        for i7 in range (n):
            daten[str(i7)]['r_end'] += delta_r
```

Die Positionen und Geschwindigkeiten aller Körper werden am Ende des Zeitschritts in die jeweiligen Listen eingefügt.

```
# Die Endpositionen und Endgeschwindigkeiten der Körper 0, 1, ..., n werden
unter dem Key 'R' und 'V' gespeichert
for i8 in range (n):
    daten[str(i8)]['R'].append(daten[str(i8)]['r_end'])
    daten[str(i8)]['V'].append(daten[str(i8)]['v_new'])
```

Die potentielle, die kinetische Energie und die Gesamtenergie werden am Ende des Zeitschritts berechnet und in die dafür erstellten Listen eingefügt.

```
# Energieberechnung
E_pot.append (potentielle_Energie())
E_kin.append (kinetische_Energie())
```

```
E_G.append (Gesamtenergie())
# Impulsberechnung
p_G.append (Gesamtimpuls())
```

Nach der *for*-Schleife ist die Berechnung abgeschlossen. Da die Geschwindigkeiten und Positionen direkt im Dictionary *daten* gespeichert sind, können sie auch ausserhalb der Funktion abgerufen werden und müssen nicht mit *return* zurückgegeben werden. Die Listen für die Energien und Impulse sind jedoch nicht im Dictionary *daten* gespeichert und werden für die weitere Verwendung mit *return* als Tuple zurückgegeben.

```
return E_pot, E_kin, E_G, p_G
```

5.1.3 Aufrufen der Berechnung durch den Leapfrog-Algorithmus

Die Funktion *leapfrog* wird aufgerufen und das zurückgegebene Tuple wird ausgepackt, sodass die Liste für die potentielle Energie *E_pot*, für die kinetische Energie *E_kin*, für die Gesamtenergie *E_G* und für den Gesamtimpuls *p_G* direkt verwendet werden kann.

```
E_pot, E_kin, E_G, p_G = leapfrog (time,time_steps)
```

5.1.4 Energie- und Impulsbilanz

Um eine Idee von der Energiebilanz der Simulation zu erhalten, wird die Gesamtenergie am Ende und die Gesamtenergie am Anfang der Berechnung ausgegeben und anschliessend deren Differenz.

```
# Energieerhaltung
print ('Energieerhaltung')
print ('Gesamtenergie (Ende), Gesamtenergie (Anfang)')
print (E_G[-1], E_G[0])
print ('ΔGesamtenergie')
print (E_G[-1] - E_G[0])
print ()
```

Analog zur Energieerhaltung wird auch für die Impulserhaltung zuerst der Gesamtimpuls am Ende und der Gesamtimpuls am Anfang der Berechnung ausgegeben. Anschliessend wird deren Differenz berechnet und ausgegeben.

```
# Impulserhaltung
print ('Impulserhaltung')
print ('Gesamtimpuls (Ende), Gesamtimpuls (Anfang)')
print (p_G[-1], p_G[0])
print ('ΔGesamtimpuls')
print (p_G[-1] - p_G[0])
```

5.1.5 Speicherung der Ergebnisse

Zunächst werden alle Positionen aller Körper in den csv-Files gespeichert. Dafür wird eine Funktion genutzt, die im File *daten_speichern* definiert wurde (vgl. 5.4). Für jeden Körper wird ein eigenes File für die Positionen erstellt. Pro Linie im csv-File sind die x-, y- und z-Koordinaten einer Position zu sehen. Anschliessend werden gleichermassen alle

Geschwindigkeiten aller Körper in neuen csv-Files gesichert. Die Namen der csv-Files werden aus der Nummer des jeweiligen Körpers und, für die Positionen, *_Positionen*, oder, für die Geschwindigkeiten *_Geschwindigkeiten* zusammengestellt. Am Ende der Namen der csv-Files wird das durch den Input Parameter *Zusatz* der Funktion *n_Körperproblem* gegebene Kürzel angehängt. (vgl. 5.1.2)

Ebenfalls wird die potentielle und die kinetische Energie im System sowie die Gesamtenergie und der Gesamtimpuls über die Zeitdauer der Berechnung gespeichert. Die Namen der dafür erstellten csv-Files lauten: *potentielle_Energie*, *kinetische_Energie*, *Gesamtenergie* und *Gesamtimpuls*. Am Ende dieser Namen wird ebenfalls das durch den Parameter *Zusatz* gegebene Kürzel angehängt.

```
# Speicherung der Positionen und Geschwindigkeiten jeder Körper im
jeweiligen csv-File
for i9 in range(n):
    ds.save_csv_array(daten[str(i9)]['R'], str(i9)+'_Positionen' + Zusatz
+'.csv')
    ds.save_csv_array(daten[str(i9)]['V'], str(i9)+'_Geschwindigkeiten' +
Zusatz +'.csv')
ds.save_csv_list(E_pot, 'potentielle_Energie' + Zusatz +'.csv')
ds.save_csv_list(E_kin, 'kinetische_Energie' + Zusatz +'.csv')
ds.save_csv_list(E_G, 'Gesamtenergie' + Zusatz +'.csv')
ds.save_csv_list(p_G, 'Gesamtimpuls' + Zusatz +'.csv')
```

5.1.6 Darstellung der Ergebnisse

Die Ergebnisse werden mit Funktionen aus der Bibliothek *matplotlib.pyplot*, welche zu Beginn des Programms dafür importiert wurde, dargestellt. (vgl. 5.1.1)

5.1.6.1 Separierung der Koordinaten der Positionen

Um die Positionen der Körper darstellen zu können, müssen die x-, y- und z-Koordinaten voneinander separiert werden. Listen für alle x-Koordinaten jedes Körpers werden im Dictionary *daten* unter dem Dictionary des jeweiligen Körpers unter dem Key *'x'* gespeichert. Die Listen für die y- und z-Koordinaten der Positionen werden gleichermassen erstellt. (vgl. 4.3.1)

```
# Erstellung der Listen für die Separierung der x-, y- und z-Koordinaten
der Positionen der Körper 0, 1, ..., n
for i10 in range (n):
    daten[str(i10)]['x'] = []
    daten[str(i10)]['y'] = []
    daten[str(i10)]['z'] = []
```

Anschliessend werden in zwei *for*-Schleifen von jeder Position jedes Körpers die Koordinaten in die jeweiligen Listen eingefügt. Dabei werden zuerst die Komponenten der Koordinaten der Positionen vom ersten Zeitschritt für alle Körper, dann die Komponenten der Koordinaten der Positionen vom zweiten Zeitschritt für alle Körper in die jeweiligen Listen eingefügt, usw. Damit Ergebnisse, wie in der Astronomie üblich, in *AE* dargestellt werden können, werden die Koordinaten beim Einfügen in die jeweiligen Listen von m in *AE* umgerechnet, indem sie durch die aus *scipy.constants* importierte Grösse *au* geteilt werden (vgl. 5.1.1).

```
# Separierung der x-, y- und z-Koordinaten der Positionen und gleichzeitige
Umwandlung der Koordinaten von m in AE
for j in range (0,len(daten['0']['R'])):
    for i11 in range (n):
        daten[str(i11)]['x'].append(daten[str(i11)]['R'][j][0]/au)
        daten[str(i11)]['y'].append(daten[str(i11)]['R'][j][1]/au)
        daten[str(i11)]['z'].append(daten[str(i11)]['R'][j][2]/au)
```

5.1.6.2 Darstellung der Positionen

Es wird zunächst der Plot initiiert. Darauf wird festgelegt, dass der Plot drei Subplots haben soll, welche in einem $1 \cdot 3$-Gitter angeordnet sein sollen. Der erste Subplot befindet sich links. Dieser stellt den Verlauf der Positionen der Körper dar. Der Name diese Plots wird aus der Anzahl verwendeter Körper und dem Zusatz *-Körperproblem* zusammengestellt. Im Fall der Simulation der Sonne und der Planeten unseres Sonnensystems lautet der Titel *9-Körperproblem*. Die Achsen werden mit *x [AE]*, *y [AE]* und *z [AE]* beschriftet. (vgl. 4.3.1)

```
# Erstellen des Plots
fig = plt.figure()
ax = fig.add_subplot(1,3,1, projection="3d")
# Titel des Subplots
ax.set_title (str(n)+'-Körperproblem')
# Achsenbeschriftungen
ax.set_xlabel ('x [AE]')
ax.set_ylabel ('y [AE]')
ax.set_zlabel ('z [AE]')
```

In der ersten *for*-Schleife werden die Bahnen der Körper mit den Positionen gezeichnet. In der zweiten *for*-Schleife, werden die Endpositionen etwas grösser dargestellt, um zu verdeutlichen, wo sich die Körper am Ende der Simulation befinden. Dabei werden sie als Punkte dargestellt, welche ihre relative Grösse repräsentieren soll. Diese relative Grösse ist im Dictionary des jeweiligen Körpers gespeichert. Zusätzlich wird Name des jeweiligen Körpers, der im Dictionary *daten* gespeichert ist, als Label für die Legende angegeben. In der dritten *for*-Schleife werden die erwarteten Endpositionen in den Plot eingegeben, falls der Parameter Vergleich der Funktion *n_Körperproblem* auf *True* gesetzt wurde.

```
# Darstellung der Bahnen der Körper
for i12 in range(n):
    ax.plot(daten[str(i12)]['x'], daten[str(i12)]['y'],
daten[str(i12)]['z'], color = daten[str(i12)]['farbe'], linewidth = 0.5)

# berechnete Endposition
for i13 in range(n):
    ax.plot(daten[str(i13)]['x'][-1], daten[str(i13)]['y'][-1],
daten[str(i13)]['z'][-1], linestyle = '', marker = '.', ms =
daten[str(i13)]['grösse'], color = daten[str(i13)]['farbe'], label =
daten[str(i13)]['name'])

# erwartete Endposition
if Vergleich:
    for i14 in range(n):
```

```
        ax.plot(daten[str(i14)]['end_r'][0], daten[str(i14)]['end_r'][1],
daten[str(i14)]['end_r'][2], linestyle = '', marker = '.', color = 'green')

ax.legend()
```

5.1.6.3 Darstellung der Energien

Der zweite Plot befindet sich in der Mitte und stellt den Verlauf der kinetischen und der potentiellen Energie, sowie der Gesamtenergie dar. Es wird zunächst ein Array *x_E* erstellt, der bei $t_0 = 0$ startet mit Zeitschritten von Δt bei $t_e = T$ aufhört. Er wird für die Werte der x-Achse beim Plot der Energieerhaltung und der Impulserhaltung verwendet.

```
x_E = np.linspace(0, int(time), len(E_pot))
x_E *= time_steps
```

Mit *fig.add_subplot (1,3,2)* wird der Subplot für die Energieerhaltung erstellt. Der Titel dieses Subplots lautet *Energieerhaltung*. Die x-Achse ist die Zeitachse. Die Zeit ist dabei durch den Parameter *Gesamtlaufdauer* definiert und wird in s angegeben. Auf der y-Achse werden die Verläufe der Energien in *J* dargestellt. Die x-Achse wird mit *Zeit [s]* und die y-Achse mit *Energie [J]* beschriftet. Mit *ax.plot* werden die Werte der Gesamtenergie, der kinetischen Energie und der potentiellen Energie im gleichen Plot eingetragen und in der Legende durch den Zusatz *label* = in der Legende des Plots aufgeführt.

```
ax = fig.add_subplot(1,3,2)
ax.set_title ('Energieerhaltung')
ax.set_xlabel ('Zeit [s]')
ax.set_ylabel ('Energie [J]')
ax.plot (x_E, E_G, label = 'Gesamtenergie')
ax.plot (x_E, E_kin, label = 'kinetische Energie')
ax.plot (x_E, E_pot, label = 'potentielle Energie')
ax.legend()
```

5.1.6.4 Darstellung des Gesamtimpulses

Bei der Darstellung des Verlaufs des Gesamtimpulses dient ebenfalls die x-Achse als Zeitachse. Wie beim Plot der Energieerhaltung wird für die Werte der x-Achse der Array *x_E* verwendet. Die y-Achse dient der Darstellung des Gesamtimpulses. Die x-Achse wird analog zur Darstellung der Energien mit *Zeit [s]* und die y-Achse wird mit *Impuls [Ns]* beschriftet.

```
ax = fig.add_subplot (1,3,3)
ax.set_title ('Impulserhaltung')
ax.set_xlabel ('Zeit [s]')
ax.set_ylabel ('Impuls [Ns]')
ax.plot (x_E, p_G, label = 'Gesamtimpuls')
ax.set_ylim(0.5 * max(p_G) , 1.5*max(p_G))
ax.legend()
```

5.1.6.5 Vergrösserte Darstellung der Positionen

Damit der Verlauf der Bahnen und die Positionen der Körper etwas besser zu erkennen sind, werden sie in einem separaten $1 \cdot 1$-Plot ein zweites Mal grösser dargestellt. Der einzige

Unterschied zu 5.1.6.2 besteht darin, dass mit der Funktion *fig.add_subplot* nur ein Subplot erstellt wurde und dass die Achsen mit proportional *ax.set_aspect ('equal')* beschriftet wurden.

Am Ende werden beide Plots in zwei Fenstern mit *plt.show()* ausgegeben.

```
fig = plt.figure()
ax = fig.add_subplot(1,1,1, projection = '3d')
ax.set_title (str(n)+'-Körperproblem')
ax.set_xlabel ('x [AE]')
ax.set_ylabel ('y [AE]')
ax.set_zlabel ('z [AE]')
ax.set_aspect ('equal')

for i15 in range(n):
    ax.plot(daten[str(i15)]['x'], daten[str(i15)]['y'],
daten[str(i15)]['z'], color = daten[str(i15)]['farbe'])

# berechnete Endposition
for i16 in range(n):
    ax.plot(daten[str(i16)]['x'][-1], daten[str(i16)]['y'][-1],
daten[str(i16)]['z'][-1], linestyle = '', marker = '.', ms =
daten[str(i16)]['grösse'], color = daten[str(i16)]['farbe'], label =
daten[str(i16)]['name'])

# erwartete Endposition
if Vergleich:
    for i17 in range(n):
        ax.plot(daten[str(i17)]['end_r'][0], daten[str(i17)]['end_r'][1],
daten[str(i17)]['end_r'][2], linestyle = '', marker = '.', color = 'green')

ax.legend()
plt.show()
```

5.2 Simulation des n-Körperproblem

Das n-Körperproblem wird durch den Aufruf der Funktion *n_Körperproblem* simuliert. Dies geschieht nicht im File *n_Körperproblem_Berechnungen* selbst, sondern der Benutzerfreundlichkeit wegen im File *Simulation_n_Körperproblem_aus_Berechnungen*. Darin befinden sich zwei Funktionen. Die erste simuliert den Spezialfall des Dreikörperproblems des gleichseitigen Dreiecks, wie er in 6.1.1 analysiert wurde. Die zweite Funktion simuliert das n-Körperproblem mit der Sonne und allen Planeten unseres Sonnensystems, wie er in 6.2.1 analysiert wurde. Um andere Fälle des n-Körperproblems zu simulieren, müsste zuerst für die verwendeten Körper Dictionary-Einträge erstellt werden, wie in 5.5 beschrieben wurde.

Um auf die Files *n_Körperproblem_Berechnungen* und *Himmelskörper_Daten* zugreifen zu können, müssen diese zunächst importiert werden.

```
from n_Körperproblem_Berechnungen import n_Körperproblem
import Himmelskörper_Daten as hd
```

5.2.1 Spezialfall des Dreikörperproblems: Gleichseitiges Dreieck

Die Funktion, welche den Spezialfall des Dreikörperproblems des Gleichseitigen Dreieck aufruft, heisst *Simulation_Lagrange_Dreieck*. Diese Funktion hat keine Input Parameter. Unten ist die Funktion als Docstring aufgeführt:

```
def Simulation_Lagrange_Dreieck ():
    '''
    Input Parameter
    None

    Returns:
    Simuliert den von Lagrange entdeckten Spezialfall des
Dreikörperproblems
    mit drei Körpern der gleichen Masse, welche zusammen ein gleichseitiges
    Dreieck bilden und sich in einer Kreisbahn um ihren gemeinsamen
    Massenmittelpunkt bewegen.
    Dieses Beispiel ist fiktiv. Die Simulation spielt sich über den
Zeitraum
    von einem Erdjahr ab.
    '''
```

In der Funktion *Simulation_Lagrange_Dreieck* wird zunächst der Dictionary *Daten* erstellt. Dem Key *'0'* im Dictionary *Daten* wird der Dictionary *data['Körper 1']* aus dem File *Himmelskörper_Daten* eingefügt. Gleichermassen wird unter dem Key *'1'* der Dictionary *data['Körper 2']* und unter dem Key *'2'* der Dictionary *data['Körper 3']* eingefügt.

```
Daten = {}
Daten['0'] = hd.data['Körper 1']
Daten['1'] = hd.data['Körper 2']
Daten['2'] = hd.data['Körper 3']
```

Anschliessend wird die Gesamtlaufdauer der Berechnung definiert und in Sekunden umgerechnet. Hier wird sie auf 1 Jahr festgelegt. Der Zeitschritt Δt für die Berechnung mit dem Leapfrog-Algorithmus wird auf $3600\ s = 1\ h$ festgelegt.

```
# Gesamtzeit für Laufdauer des Programms
Zeit = 3600 * 24 * 365.242199 * 1
# Zeitschritt für Leapfrog-Algorithmus
Zeitschritt = 3600
```

Dann wird die Funktion *n_Körperproblem* mit allen notwendigen Parametern aufgerufen. Es wird der Dictionary *daten* für den Parameter *daten*, den Integer 3 für den Parameter *n*, die Variable *Zeit* für den Parameter *Gesamtlaufdauer*, die Variable *Zeitschritt* für den Parameter *delta_T* eingegeben. Da der Massenmittelpunkt den Koordinatenursprung bildet und sich alle Körper sichtbar bewegen, wird der Parameter *Fixierung* auf *False* gesetzt. Da es keine Ephemeriden zum Vergleich dieses spezifischen Beispiels gibt (die Situation ist erfunden), wird für den Parameter *Vergleich* ebenfalls *False* angegeben. Für den Parameter *Zusatz*, welcher für die Speicherung der berechneten Positionen, Geschwindigkeiten, Gesamtenergien und des berechneten Gesamtimpulses verwendet wird, wird der String *'_Lagrange_Dreieck'* eingefügt.

```
# daten, n, Gesamtlaufdauer, delta_T, Fixierung, Vergleich, Zusatz
n_Körperproblem(Daten, 3, Zeit, Zeitschritt, False, False,
'_Lagrange_Dreieck')
```

Sobald die Berechnungen fertig abgelaufen und die erhaltenen Positionen, Geschwindigkeiten, Energien und Impulse in den jeweiligen csv-Files gespeichert sind, werden die folgenden zwei Plots ausgegeben. Auf dem ersten sieht man die Bewegungen der Körper und die Energie- und Impulserhaltung (vgl. Abbildung 5). Auf dem zweiten Plot, welcher in einem separaten Fenster dargestellt ist, sieht man die Bewegungen der Körper noch einmal vergrössert. (vgl. Abbildung 7)

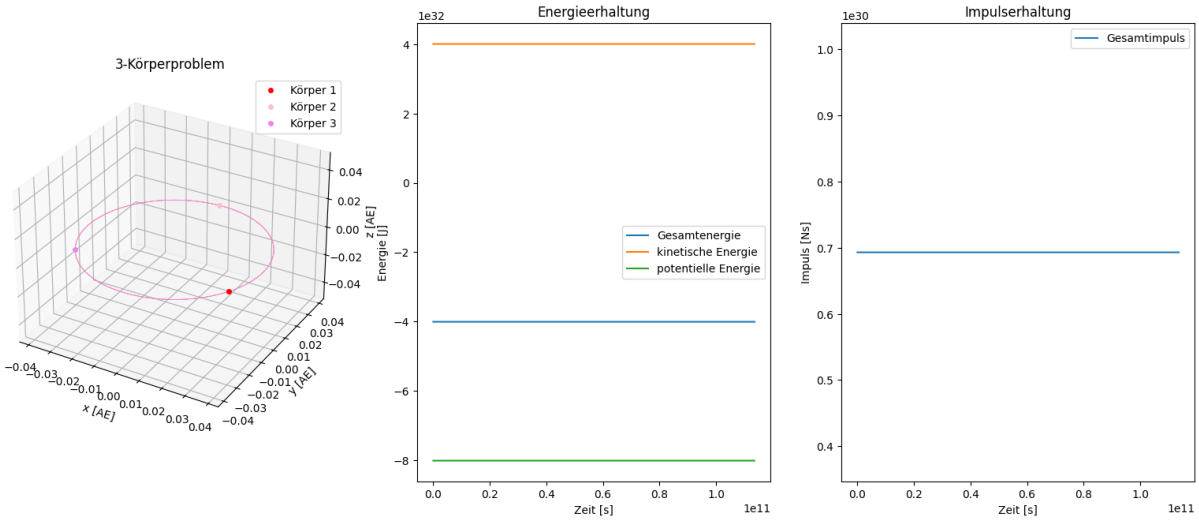


Abbildung 5: 3 Subplots des Dreikörperproblems (Gleichseitiges Dreieck)

5.2.2 n-Körperproblem mit Sonne und Planeten unseres Sonnensystems

Die Simulation für das n-Körperproblem mit der Sonne und allem Planeten unseres Sonnensystems erfolgt in der Funktion *Simulation_Sonne_und_Planeten*. Diese Funktion ruft die Funktion so auf, wie sie in 6.2.1 analysiert wurde. Diese Funktion erfordert keine Input Parameter und ist wie folgt definiert:

```
def Simulation_Sonne_und_Planeten ():
    '''
    Input Parameter
    None

    Returns:
    Simuliert das n-Körperproblem mit der Sonne und allen Planeten unseres
Sonnesystems
    Beginn der Simulation: 23.09.2015
    Ende der Simulation: 23.09.2025
    '''
```

Wie schon bei 5.2.1 wird zuerst der Dictionary *Daten* initiiert. Anschliessend werden die Dictionaries der verwendeten Körper unter den Keys *'0'* bis *'8'* eingefügt. Die Nummerierung folgt dabei dem aufsteigenden Abstand von der Sonne. So wird unter dem Key *'0'* der Dictionary der 'Sonne', unter dem Key *'1'* der Dictionary des Merkurs, unter dem Key *'2'* der Dictionary der Venus, unter dem Key *'3'* der Dictionary der Erde, usw. eingefügt.

```
Daten = {}
Daten['0'] = hd.data['Sonne']
Daten['1'] = hd.data['Merkur']
Daten['2'] = hd.data['Venus']
Daten['3'] = hd.data['Erde']
Daten['4'] = hd.data['Mars']
Daten['5'] = hd.data['Jupiter']
Daten['6'] = hd.data['Saturn']
Daten['7'] = hd.data['Uranus']
Daten['8'] = hd.data['Neptun']
```

Anschliessend wird die Zeit, über welche die Berechnungen durchgeführt werden sollen, definiert und in Sekunden umgerechnet. Im vorliegenden Fall wird die Zeit auf 10 Jahre festgelegt. Der Zeitschritt Δt für die Berechnung mit dem Leapfrog-Algorithmus wird auf $3600s = 1h$ festgelegt.

```
# Gesamtzeit für Laufdauer des Programms
Zeit = 3600 * 24 * 365.242199 * 10
# Zeitschritt für Leapfrog-Algorithmus
Zeitschritt = 3600
```

Mit den oben definierten Grössen wird die Funktion *n_Körperproblem* aufgerufen. Für Parameter *daten* wird der Dictionary *daten* eingefügt, für den Parameter *n* der Integer *9*, für die *Gesamtlaufdauer* die Variable *Zeit*, für *delta_T* die Variable *Zeitschritt*. Für den Parameter *Fixierung* wird der Boolean Value *True* eingegeben, da die Ephemeriden die Sonne als Ursprung des Koordinatensystems definieren. Der Parameter *Vergleich* wird ebenfalls auf *True* gesetzt, da es Ephemeriden zur Endposition gibt, mit denen die berechneten Positionen verglichen werden sollen. Für den letzten Parameter *Zusatz* wird der String *'_Sonnensystem'* eingefügt.

```
# daten, n, Gesamtlaufdauer, delta_T, Fixierung, Vergleich, Zusatz
n_Körperproblem(Daten, 9, Zeit, Zeitschritt, True, True, '_Sonnensystem')
```

Da die Gesamtlaufdauer der Simulation 10 Jahre beträgt und der Verlauf der Positionen für 9 Körper berechnet werden muss, müssen sehr viele Berechnungen durchgeführt werden, wodurch die Simulation sehr lange dauert. Die berechneten 87660 Positionen und Geschwindigkeiten jedes Körpers werden danach in csv-Files gesichert, damit die Berechnungen nur einmal durchgeführt werden müssen und anschliessend aufgrund der erhaltenen Daten die Simulation ein zweites Mal deutlich schneller dargestellt werden kann. Die Speicherung der 87660 Positionen und Geschwindigkeiten pro Körper verlängert allerdings die Laufzeit des Programms. Die Laufzeit für die Simulation beträgt ungefähr $2\,min,\ 18\,sek$.

Nach dieser Laufzeit werden die wie schon in 5.2.1 die Resultate in zwei Plots dargestellt. Auf dem ersten sieht man die Bewegungen der Körper und die Energie- und Impulserhaltung (vgl. Abbildung 6). Auf dem zweiten Plot, welcher in einem separaten Fenster dargestellt ist, sieht man die Bewegungen der Körper noch einmal vergrössert. (vgl. Abbildung 9)

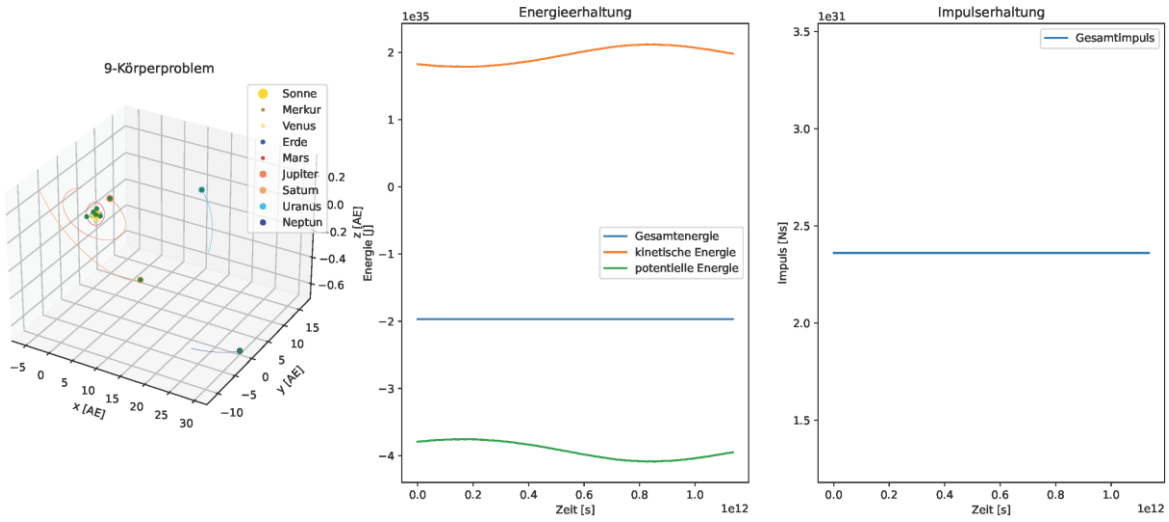


Abbildung 6: 3 Subplots des 9-Körperproblems (Sonnensystems)

5.3 Animierte Simulation

Damit eine animierte Simulation ohne die langen Berechnungen gezeigt werden kann, werden die in den csv-Files gespeicherten Daten verwendet. Genau das geschieht im File *Animierte_Simulation_aus_csv*. Dieses Programm funktioniert nur, wenn die Berechnungen bereits einmal durchgeführt worden sind und die Positionen der Körper für die Simulation in csv-Files gespeichert vorliegen. Falls dies nicht der Fall ist, muss zuerst das Programm im File *Simulation_n_Körperproblem_aus_Berechnungen* vollständig laufen gelassen werden, und die Plots müssen manuell geschlossen werden. (vgl. 5.2)

5.3.1 Importe

Als erstes werden die nötigen Bibliotheken importiert. Dies wären *numpy* für die Arrays sowie *matplotlib.pyplot* für die Darstellung der Simulation. Ausserdem wird die Konstante *au* für die Umwandlung von *m* in *AE* aus der Bibliothek *scipy.constants* importiert. Die Bibliothek *csv* ermöglicht das Einlesen der Daten aus den csv-Files und die Bibliothek *Himmelskörper_Daten* liefert die Farben, Namen und relativen Grössen der Körper.

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.constants import au
import csv
import Himmelskörper_Daten as hd
```

5.3.2 Animation

Die Funktion *Animation* verlangt 4 Parameter. Der erste Parameter ist der Dictionary *daten* mit den Unterdictionaries für jeden Körper, in welchen (mindestens) die Farbe (Key ‘farbe’),

der Name (Key ‘name’) und die relative Grösse (Key ‘grösse’) gespeichert sind. Dazu ist es wichtig, dass derjenige Körper, der mit dem Key *str(n-1)* gespeichert ist, die grösste Distanz zum Ursprung des Koordinatensystems aufweist. Der Parameter *n* legt die Anzahl verwendeter Körper fest. Der Parameter *Zusatz* bestimmt das Kürzel am Ende des Namens des csv-Files und der Parameter *sl* definiert die Länge der darzustellenden absolvierten Bahn des Körpers in Anzahl Positionen im csv-File geteilt durch 100.

```
def Animation(daten, n, Zusatz, sl):
    '''
    Input Parameter:
    daten:
        Dictionary mit Daten aller n Himmelskörper
            Für jeden Himmelskörper eigener Dictionary in daten mit Key
'0', '1', ... str(n-1) mit:
                Farbe des Körpers (Key 'farbe')
                Relative Grösse des Körpers (Key 'grösse')
                Name des Körpers (Key 'name')
    n:
        Anzahl verwendeter Körper
    Zusatz:
        Kürzel am Ende des Names des csv-Files
    sl:
        Länge der dargestellten, bereits absolvierten Bahn in Anzahl
gespeicherter Positionen

    Returns:
    animierte Simulation'''
```

Zunächst wird in einer *for*-Schleife das csv-File mit den Positionen von jedem Körper geöffnet, es werden im Dictionary des jeweiligen Körpers Listen für die x-, y- und z-Koordinaten erstellt. Anschliessend werden die Komponenten der Koordinaten von jeder Position in einer zweiten *for*-Schleife in die jeweilige Liste eingefügt und in *AE* umgewandelt. Falls ein Error entsteht, weil das File nicht gefunden wurde, muss allenfalls sein Dateipfad angepasst werden. Der aktuelle Dateipfad greift auf die Files auf dem USB-Stick MIKUELLING im Ordner *Files* zu.

```
for i1 in range (n):
    # öffnen des Files des Körpers mit dem Key str(i1)
    with open('D:/Files/' + str(i1) + '_Positionen'+ Zusatz +'.csv') as
csvfile:
        daten[str(i1)]['reader'] = csv.reader(csvfile, delimiter=',',
quotechar='"')

        # Erstellen der Listen für die x-, y- und z-Komponenten der
Koordinaten der Positionen
        daten[str(i1)]['x'] = []
        daten[str(i1)]['y'] = []
        daten[str(i1)]['z'] = []

        # Einfügen der Komponenten der Koordinaten in die jeweiligen Listen
und Umwandlung in AE
        for row in daten[str(i1)]['reader']:
```

```
            daten[str(i1)]['x'].append(float(row[0])/au)
            daten[str(i1)]['y'].append(float(row[1])/au)
            daten[str(i1)]['z'].append(float(row[2])/au)
```

Dann wird der Plot initiiert und seine Grenzen für die Animation berechnet. Anschliessend wird in einer *for*-Schleife jede 100ste gespeicherte Position in der Animation dargestellt. Es wird die Grösse des Plots festgelegt. Dann wird festgelegt, dass die Achsen gleich skaliert sein müssen. Die Achsen werden beschriftet und in einer weiteren *for*-Schleife werden die (sich bewegenden) Punkte aller Körper dargestellt. Dann wird der Index des letzten darzustellenden Elements festgelegt, und für jeden Körper wird in der nächsten *for*-Schleife die bereits absolvierte Teilstrecke seiner Bahn, deren Länge mit *sl* bestimmt wurde, als Schweif dargestellt. Mit *plt.pause* wird der jeweilige Frame für 0.001 *s* dargestellt bevor er mit *plt.cla* geschlossen wird. (vgl. 4.3.2)

```
for q in range (0, len(daten['0']['x']), 100):
    # Grösse des Plots festlegen
    ax.set_ylim (-lim, lim)
    ax.set_xlim (-lim, lim)
    ax.set_zlim (-lim, lim)

    # gleiche Skalierung der Achsen festlegen
    ax.set_aspect ('equal')

    # Achsenbeschriftungen
    ax.set_xlabel ('x [AE]')
    ax.set_ylabel ('y [AE]')
    ax.set_zlabel ('z [AE]')

    # Darstellung der bewegenden Punkte
    for i2 in range (n):
        ax.plot(daten[str(i2)]['x'][q], daten[str(i2)]['y'][q],
daten[str(i2)]['z'][q], marker = '.', color = daten[str(i2)]['farbe'],
label = daten[str(i2)]['name'], ms = daten[str(i2)]['grösse'])

    # Index für den letzten Punkt der darzustellenden Teilstrecke der
bereits absolvierten Bahn
    if q - sl < 0:
        p = 0
    else:
        p = q - sl

    # Darstellung der Teilstrecke der bereits absolvierten Bahn
    for i3 in range (n):
        ax.plot(daten[str(i3)]['x'][p:q], daten[str(i3)]['y'][p:q],
daten[str(i3)]['z'][p:q], color = daten[str(i3)]['farbe'])

    ax.legend()

    # Darstellung des Frames
    plt.pause (0.001)
    # Schliessen des Frames
    plt.cla()
```

5.3.3 Spezialfall des Dreikörperproblem: Gleichseitiges Dreieck

Die Funktion *Animation_Lagrange_Dreieck* ruft die Funktion *Animation* so auf, dass der Spezialfall des Dreikörperproblem des gleichseitigen Dreiecks als animierte Simulation dargestellt wird. Die Funktion *Animation_Lagrange_Dreieck* verlangt keine Parameter. Sie ist unten als Docstring aufgeführt.

```
def Animation_Lagrange_Dreieck():
    '''
    Input Parameter
    None

    Returns:
    Simuliert den von Lagrange entdeckten Spezialfall des
Dreikörperproblems
    mit drei Körpern der gleichen Masse, welche zusammen ein gleichseitiges
    Dreieck bilden und sich in einer Kreisbahn um ihren gemeinsamen
    Massenmittelpunkt bewegen.
    Dieses Beispiel ist fiktiv. Die Simulation spielt sich über den
Zeitraum
    von einem Erdjahr ab.
    '''
```

In der Funktion *Animation_Lagrange_Dreieck* wird der Dictionary *Daten* erstellt und die Daten für den Körper 1, den Körper 2 und Körper aus dem Dictionary *data* im File *Himmelskörper_Daten* werden unter den Keys *'0'*, *'1'* und *'2'* eingefügt.

```
Daten = {}
Daten['0'] = hd.data['Körper 1']
Daten['1'] = hd.data['Körper 2']
Daten['2'] = hd.data['Körper 3']
```

Anschliessend wird die Funktion Animation mit dem Dictionary *Daten* für den Parameter *daten*, dem Integer *3* für den Parameter *n*, dem String *'_Lagrange_Dreieck'* für den Parameter *Zusatz* und dem Integer *2900* für den Parameter *sl*.

```
Animation(Daten, 3, '_Lagrange_Dreieck', 2900)
```

5.4 Daten speichern

5.4.1 Importe

Die Bibliothek *csv* wird importiert. Diese wird für das Schreiben eines csv-Files verwendet, wodurch Daten gesichert werden können.

5.4.2 Sichern von Listen mit Arrays

Die Funktion *save_csv_array* sichert eine Liste von Arrays. Dabei wird jeder Array auf einer neuen Zeile im csv-File eingefügt. Die Funktion verlangt zwei Parameter: *arr_lis*, eine Liste von Arrays und *nam*, der Name des erstellten csv-Files. Die Funktion gibt keinen Wert zurück.

Zuerst wird ein File mit dem Namen, der unter *nam* angegeben ist, geöffnet. Anschliessend werden die Arrays mit dem *csv.writer* und der Funktion *writerows* Zeile für Zeile ins csv-File eingefügt. Zuletzt wird das fertige File geschlossen. Das File wird auf dem Stick MIKUELLING im Ordner *Files* gespeichert. Falls ein Error entsteht, muss allenfalls der Dateipfad angepasst werden.

```
def save_csv_array(arr_lis, nam):
    '''
    Input Parameters:
    arr_lis: Liste mit Arrays, welche in einem csv file gespeichert werden
soll
    nam: Name des csv files

    Returns:
    speichert die Liste arr_lis in csv file'''

    # neues File mit Namen 'nam' öffnen
    csvfile = open('D:/Files/' + nam, 'w', newline='', encoding='utf-8')
    c = csv.writer(csvfile)

    # Liste ins File einfügen
    c.writerows(arr_lis)

    # File schliessen und speichern
    csvfile.close()
```

5.4.3 Sichern von Listen mit Floats

Die zweite Funktion des Files *daten_speichern* speichert eine Liste mit Floats. Die Funktion *save_csv_list* verlangt zwei Parameter: *lis*, eine Liste von Floats und ebenfalls *nam*, der Name des zu erstellenden csv-Files. Auch diese Funktion gibt keinen Wert zurück.

Es wird analog zu *save_csv_array* zuerst ein csv-File mit dem Namen *nam* geöffnet und mit dem *csv.writer* die Werte ins File eingefügt. Im Unterschied zu *save_csv_array* wird jedes Element der Liste einzeln mit der Funktion *writerow* ins csv-File eingefügt. So landen nicht alle Werte auf derselben Zeile, sondern jeder Wert wird auf einer eigenen Zeile eingefügt. Das File wird ebenfalls gemäss dem aktuellen Dateipfad auf dem USB-Stick MIKUELLING im Ordner *Files* gespeichert.

```
def save_csv_list(lis, nam):
    '''
    Input Parameter:
    lis: Liste mit floats, welche in einem csv file gespeichert werden soll
    nam: Name des csv files

    Returns:
    speichert die Liste lis in csv file'''

    # neues File mit Namen 'nam' öffnen
    csvfile = open('D:/Files/' + nam, 'w', newline='', encoding='utf-8')
    c = csv.writer(csvfile)
```

```
    # Liste ins File einfügen
    for elem in lis:
        c.writerow([elem])

    # File schliessen und speichern
    csvfile.close()
```

5.5 Daten der Himmelskörper

Alle Daten für die verwendeten Himmelskörper befinden sich im File *Himmelskörper_Daten.*

Darin werden zunächst die Bibliothek *numpy* für die Arrays und Wurzelfunktion und aus der Bibliothek *scipy.constants* die Konstanten *au* und *G* für die Vektorberechnungen für die Anfangsbedingungen des Spezialfalls des Dreikörperproblems des gleichseitigen Dreiecks importiert.

```
import numpy as np
from scipy.constants import au, G
```

Dann wird der Dictionary *data* initiiert.

```
data = {}
```

Für jeden Körper wird im Dictionary *data* ein weiterer Dictionary erstellt. Darin gespeichert werden die Masse, die Startposition, die Startgeschwindigkeit, der Name, die relative Grösse und die Farbe des Himmelskörpers und die erwartete Endposition.

Die Masse wird unter dem Key *'m'* in kg, die Startposition unter dem Key *'r'* in m und die Startgeschwindigkeit unter dem Key *'v'* in m/s angegeben. Unter dem Key *'name'* ist der deutsche Name des Himmelskörpers vermerkt, die Farbe des Planeten ist unter dem Key *'farbe'* zu finden. Die relativen Grössen der Himmelskörper sind im Dictionary des jeweiligen Körpers unter dem Key *'grösse'* aufgelistet. Dabei wurde darauf geachtet, dass die relative Grösse mit grösserer Masse des Körpers zunimmt und mit kleinerer Masse abnimmt. Die relativen Grössen sind jedoch nicht proportional auf- bzw. absteigend und sind nicht auf die Abstände der Körper abgestimmt, sondern aus Gründen der Sichtbarkeit auf dem Plot deutlich grösser, als sie eigentlich dargestellt werden müssten. Als letztes Element ist die Position der Körper unter dem Key *'r_end'* in *AE* angegeben. Diese dient zum Vergleich der berechneten und der beobachteten Position und damit zur Analyse der Simulation.

5.5.1 Daten für den Spezialfall des Dreikörperproblems: Gleichseitiges Dreieck

Die Daten für die drei identischen Körper, welche im Spezialfall des Dreikörperproblems des gleichseitigen Dreiecks verwendet werden, werden zuerst in den Dictionary *data* eingefügt. Die drei Körper heissen *Körper 1*, *Körper 2* und *Körper 3*. Diese Namen dienen als Key für die Dictionaries der drei Körper im Dictionary *data*. Sie haben im Dictionary alle die relative Grösse 8 und werden im Plot rot, pink und violett eingezeichnet.

Es wird zuerst die Masse der Körper definiert. Sie ist separat als Variable definiert, damit sie leichter für alle Körper verändert werden kann.

```
# Masse der Körper
```

```
m = 2e26
```

Anschliessend wird mit der Variable *s* die Seitenlänge des gleichseitigen Dreiecks definiert, welche die drei Körper bilden.

```
# Seitenlänge des Dreiecks, welches die drei Körper bilden
s = 1e10
```

Es folgen weitere Distanzen in diesem Dreieck, abhängig von der eben definierten Seitenlänge *s*, welche für die Bestimmung der Anfangspositionen als Vektoren der Körper verwendet werden: (vgl. 3.2.2.2, 6.1.1.1)

```
# Inkreisradius
b = (np.sqrt(3)/6) * s
# Umkreisradius
r = (np.sqrt(3)/3) * s
# Halbe Seitenlänge
s_2 = s/2
```

Anschliessend wird der Betrag der Geschwindigkeit der Körper berechnet: (vgl. 3.2.2.3, 6.1.1.1.2)

```
# Betrag der Geschwindigkeit der drei Körper
v = np.sqrt(G*3*m/(s**3))*r
```

Es folgen die konkreten Einträge für die drei Körper. Der Körper 1 ist rot. Der Vektor der Anfangsposition $\overrightarrow{r_1}$ und der Anfangsgeschwindigkeit $\overrightarrow{v_1}$ des Körpers 1 ist (vgl. 6.1.1.1.1, 6.1.1.1.2):

$$\overrightarrow{r_1} = \begin{pmatrix} s/2 \\ -b \\ 0 \end{pmatrix}, \qquad \overrightarrow{v_1} = \begin{pmatrix} \cos(60°) \cdot v \\ \sin(60°) \cdot v \\ 0 \end{pmatrix}$$

Der gesamte Eintrag für den Körper 1 sieht daher folgendermassen aus:

```
# Eintrag für den Körper 1
data['Körper 1'] = {}
# Masse
data['Körper 1']['m'] = m
# Geschwindigkeit (in x,y,z Richtung (kartesische Koordinaten) als
dreidimensionaler Array) in m/s
data['Körper 1']['v'] = np.array([cos60*v, sin60*v, 0])
# Position (in x,y,z Richtung (kartesische Koordinaten) als
dreidimensionaler Array) in m
data['Körper 1']['r'] = np.array([s_2, -b, 0])
# Name des Himmelskörpers
data['Körper 1']['name'] = 'Körper 1'
# Farbe des Himmelskörpers
data['Körper 1']['farbe'] = 'red'
# Relative Grösse des Himmelskörpers
data['Körper 1']['grösse'] = 8
```

Der Körper 2 ist im Plot pink eingezeichnet. Für den Körper 2 ist der Vektor der Startposition und der Vektor der Startgeschwindigkeit folgendermassen festgelegt: (vgl. 6.1.1.1.1, 6.1.1.1.2)

$$\overrightarrow{r_2} = \begin{pmatrix} 0 \\ r \\ 0 \end{pmatrix}, \qquad \overrightarrow{v_2} = \begin{pmatrix} -v \\ 0 \\ 0 \end{pmatrix}$$

Damit lautet der Eintrag für den Körper 2 folgendermassen.

```
# Eintrag für den Körper 2
data['Körper 2'] = {}
# Masse
data['Körper 2']['m'] = m
# Geschwindigkeit (in x,y,z Richtung (kartesische Koordinaten) als
dreidimensionaler Array) in m/s
data['Körper 2']['v'] = np.array([-v, 0, 0])
# Position (in x,y,z Richtung (kartesische Koordinaten) als
dreidimensionaler Array) in m
data['Körper 2']['r'] = np.array([0, r, 0])
# Name des Himmelskörpers
data['Körper 2']['name'] = 'Körper 2'
# Farbe des Himmelskörpers
data['Körper 2']['farbe'] = 'pink'
# Relative Grösse des Himmelskörpers
data['Körper 2']['grösse'] = 8
```

Der Körper 3 wird violett eingezeichnet. Der Vektor der Startposition und der Vektor der Startgeschwindigkeit des Körpers 3 lauten: (vgl. 6.1.1.1.1, 6.1.1.1.2)

$$\overrightarrow{r_3} = \begin{pmatrix} -s/2 \\ -b \\ 0 \end{pmatrix}, \qquad \overrightarrow{v_3} = \begin{pmatrix} \cos(60°) \cdot v \\ -\sin(60°) \cdot v \\ 0 \end{pmatrix}$$

Mit diesen Daten wird der Eintrag für den Körper 3 in den Dictionary wie folgt erstellt:

```
# Eintrag für den Körper 3
data['Körper 3'] = {}
# Masse
data['Körper 3']['m'] = m
# Geschwindigkeit (in x,y,z Richtung (kartesische Koordinaten) als
dreidimensionaler Array) in m/s
data['Körper 3']['v'] = np.array([cos60*v, -sin60*v, 0])
# Position (in x,y,z Richtung (kartesische Koordinaten) als
dreidimensionaler Array) in m
data['Körper 3']['r'] = np.array([-s_2, -b, 0])
# Name des Himmelskörpers
data['Körper 3']['name'] = 'Körper 3'
# Farbe des Himmelskörpers
data['Körper 3']['farbe'] = 'violet'
# Relative Grösse des Himmelskörpers
data['Körper 3']['grösse'] = 8
```

5.5.2 Daten für das n-Körperproblem

Im zweiten Teil des Files *Himmelskörper_Daten* werden die Dictionaries für die in der Simulation zum n-Körperproblem verwendeten Körper erstellt (vgl. 6.2.1). Der Key zu den jeweiligen Dictionaries ist der deutsche Name des Himmelskörpers, also *'Sonne'*, *'Merkur'*, *'Venus'*, *'Erde'*, *'Mars'*, *'Jupiter'*, *'Saturn'*, *'Uranus'* und *'Neptun'*.

Die unter dem Key *'m'* gespeicherte Masse, die Startposition (Key *'r'*) und die Startgeschwindigkeit (Key *'v'*) wurde der *Horizons Web Application* der NASA entnommen (vgl. 6.2.1.1). Die Startpositionen und Startgeschwindigkeiten sind in der Tabelle 8 aufgelistet und die Massen der Körper in der Tabelle 15 im Anhang.

Als Endposition im Dictionary unter dem Key *'r_end'* wird die Position der Körper nach 10 Jahren, also am 23.09.2025 um 00:00 Uhr, gemäss den Ephemeriden der *Horizons Web Application* der NASA (vgl. 6.2.1.1) angegeben. Sie sind in der Tabelle 9 aufgelistet.

Die Farben der Körper sind unter dem Key *'farbe'* gespeichert. Sie wurden so evaluiert, dass die Farbe der echten Farbe des Planeten möglichst nahekommt, aber die Farben der verschiedenen Planeten sich einander nicht zu sehr ähneln (vgl. 4.3.1). Die Namen, Farben und Grössen der Himmelskörper sind in der Tabelle 16 im Anhang aufgelistet.

Unten zu sehen ist der Eintrag des Merkurs als Beispiel. Alle Einträge können in einem Ausdruck des Files *Himmelskörper_Daten*, welches sich im Anhang befindet, angesehen werden.

```
# Eintrag für den Merkur
data['Merkur'] = {}
# Masse
data['Merkur']['m'] = 3.302e23
# Geschwindigkeit am 23.09.2015 um 00:00 (in x,y,z Richtung (kartesische
Koordinaten) als dreidimensionaler Array) in m/s
data['Merkur']['v'] = np.array([1.248058735416839E+01,
4.567438146577087E+01,  2.587026467585176E+00]) *1000
# Position am 23.09.2015 um 00:00 (in x,y,z Richtung (kartesische
Koordinaten) als dreidimensionaler Array) in m
data['Merkur']['r'] = np.array([5.178819007555284E+07, -
2.622832353990256E+07, -6.894403152705170E+06]) *1000
# Name des Himmelskörpers
data['Merkur']['name'] = 'Merkur'
# Farbe des Himmelskörpers
data['Merkur']['farbe'] = 'darkgoldenrod'
# Relative Grösse des Himmelskörpers
data['Merkur']['grösse'] = 4
# Endposition gemäss Ephemeride vom 23.09.2025 um 00:00
data['Merkur']['end_r'] = np.array([-5.698223562961169E+07 *1000/au, -
2.817902375598282E+07 *1000/au, 2.923546584427991E+06 *1000/au])
```

6 Analyse der Simulationen

6.1 Dreikörperproblem

Für die Simulation des Dreikörperproblems wird das Programm *n_Körperproblem_Berechnungen* verwendet. Beim Aufrufen der Funktion wird für den Parameter *n* der Integer 3 eingesetzt. Für den Dictionary *daten*, wird der aus dem File *Himmelskörper_Daten* importierte Dictionary verwendet. (vgl. 5.2.1)

6.1.1 Spezialfall: Gleichseitiges Dreieck

6.1.1.1 Startbedingungen

Wie bereits in 3.2.2 beschrieben, werden für den Spezialfall gleichseitiges Dreieck drei Körper der gleichen Masse an die Ecken eines gleichseitigen Dreiecks platziert. Für die Distanz s zwischen den Körpern wurde der Wert $s = 10^{10}\ m\ \approx\ 0.0668\ AE$ gewählt. Die Massen m_1, m_2, m_3 der drei verwendeten Körper sind alle gleich gross und betragen $2 \cdot 10^{26}\ kg$. Die Simulation läuft über einen Zeitraum von einem (tropischen) Jahr, also $3600 \cdot 24 \cdot 365.242199\ s = 31556925.99\ s$. Der Zeitschritt Δt für die numerische Integration mit dem Leapfrog-Algorithmus beträgt 1 Stunde, also $3600\ s$.

6.1.1.1.1 Anfangspositionen

Zu Beginn der Simulation befinden sich die Körper in einem gleichseitigen Dreieck, dessen untere Seite parallel zur x-Achse ausgerichtet ist. Der Ursprung des Koordinatensystems liegt beim Massenmittelpunkt des Systems, welcher gleichzeitig auch der Schwerpunkt des gleichseitigen Dreiecks ist. Da alle drei Körper in einer Ebene liegen, wird für die z-Komponente der Positionen der Körper *0* eingesetzt. Mit der in 6.1.1.1 bestimmten Distanz zwischen den Körpern (= Seitenlänge des Gleichseitigen Dreiecks) $s = 10^{10}\ m$ sowie

$$r = \frac{\sqrt{3}}{3} s$$

und

$$b = \frac{\sqrt{3}}{6} s$$

können die Ortsvektoren zu den Positionen der drei Körper wie folgt ausgedrückt werden (vgl. 3.2.2.2):

$$\overrightarrow{r_1} = \begin{pmatrix} s/2 \\ -b \\ 0 \end{pmatrix} = \begin{pmatrix} 5 \cdot 10^9\ m \\ -2.8867513459481287 \cdot 10^9\ m \\ 0 \end{pmatrix}$$

$$\overrightarrow{r_2} = \begin{pmatrix} 0 \\ r \\ 0 \end{pmatrix} = \begin{pmatrix} 0 \\ 5.773502691896258 \cdot 10^9\ m \\ 0 \end{pmatrix}$$

$$\overrightarrow{r_3} = \begin{pmatrix} -s/2 \\ -b \\ 0 \end{pmatrix} = \begin{pmatrix} -5 \cdot 10^9\ m \\ -2.8867513459481287 \cdot 10^9\ m \\ 0 \end{pmatrix}$$

6.1.1.1.2 Anfangsgeschwindigkeiten

Der Betrag der Startgeschwindigkeiten der drei Körper ist gleich gross und beträgt für alle drei Körper:

$$v = \sqrt{\frac{G\ (m_1 + m_2 + m_3)}{s^3} r} = 1155.3614153155713\ {}^{m}/_{s}$$

Dieser Wert wird auf die x-, y- und z-Komponente des Geschwindigkeitsvektors aufgeteilt. Die z-Komponente ist bei allen Körpern *0*, weil sie in einer Ebene liegen. Somit erhält man für Körper 1 $\overrightarrow{v_1}$, für Körper 2 $\overrightarrow{v_2}$ und für Körper 3 $\overrightarrow{v_3}$: (vgl. 3.2.2.3 3.2.2.3)

$$\overrightarrow{v_1} = \begin{pmatrix} \cos(60°) \cdot v \\ \sin(60°) \cdot v \\ 0 \end{pmatrix} = \begin{pmatrix} 577.68070766\ {}^{m}/_{s} \\ 1000.57233622\ {}^{m}/_{s} \\ 0 \end{pmatrix}$$

$$\overrightarrow{v_2} = \begin{pmatrix} -v \\ 0 \\ 0 \end{pmatrix} = \begin{pmatrix} -1155.3614153155713\ {}^{m}/_{s} \\ 0 \\ 0 \end{pmatrix}$$

$$\overrightarrow{v_3} = \begin{pmatrix} \cos(60°) \cdot v \\ -\sin(60°) \cdot v \\ 0 \end{pmatrix} = \begin{pmatrix} 577.68070766\ {}^{m}/_{s} \\ -1000.57233622\ {}^{m}/_{s} \\ 0 \end{pmatrix}$$

6.1.1.2 Berechnete Endpositionen

Die Körper befinden sich am Schluss der Simulation in einem scheinbar gleichseitigen Dreieck. Ihre Bahnen liegen perfekt aufeinander und formen einen Kreis. (vgl. Abbildung 7)

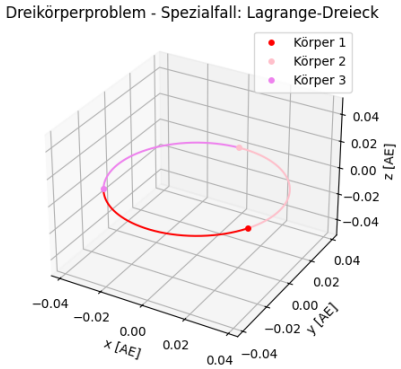


Abbildung 7: Spezialfall des Dreikörperproblem: Gleichseitiges Dreieck

Da bei allen Positionen aller Körper die z-Koordinate 0 ist, kann auch ein zweidimensionaler Plot für diesen Spezialfall des Dreikörperproblems angefertigt werden. Darauf ist noch einmal besser zu erkennen, dass die Bahnen der drei Körper einen perfekten Kreis bilden.

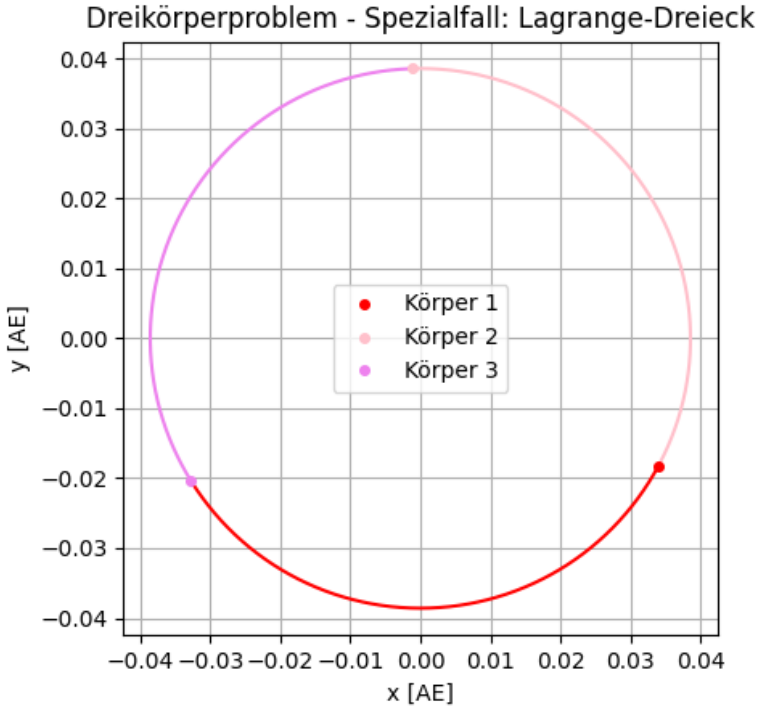


Abbildung 8: Spezialfall des Dreikörperproblem: Gleichseitiges Dreieck in 2D

Die Körper befinden sich nach der Simulation, welche über ein Jahr abgelaufen ist, sehr nahe an den Anfangspositionen. Dies ist sowohl in der Abbildung 8 wie auch in der untenstehenden Tabelle sichtbar, in welcher die Endpositionen der drei Körper und ihre jeweiligen Startpositionen aufgelistet sind. (vgl. Tabelle 1)

Körper	**Endposition** $([\vec{r}] = \boldsymbol{m})$	**Startposition** $([\vec{r}] = \boldsymbol{m})$
Körper 1	$\vec{r} = \begin{pmatrix} 5.089653094185562 \cdot 10^9 \\ -2.725576033865409 \cdot 10^9 \\ 0.0 \end{pmatrix}$	$\vec{r} = \begin{pmatrix} 5 \cdot 10^9 \\ -2.8867513459481287 \cdot 10^9 \\ 0 \end{pmatrix}$
Körper 2	$\vec{r} = \begin{pmatrix} -1.8440846181929427 \cdot 10^8 \\ 5.770556892948236 \cdot 10^9 \\ 0.0 \end{pmatrix}$	$\vec{r} = \begin{pmatrix} 0 \\ 5.773502691896258 \cdot 10^9 \\ 0 \end{pmatrix}$
Körper 3	$\vec{r} = \begin{pmatrix} -4.905244632.366352 \cdot 10^9 \\ -3.0449808590830455 \cdot 10^8 \\ 0.0 \end{pmatrix}$	$\vec{r} = \begin{pmatrix} -5 \cdot 10^9 \\ -2.8867513459481287 \cdot 10^9 \\ 0 \end{pmatrix}$

Tabelle 1: Vergleich der Endpositionen und ihrer Startpositionen

6.1.1.3 Berechnete Endgeschwindigkeiten

Bereits bei den berechneten Endpositionen wurde festgestellt, dass diese den Startpositionen sehr nahekommen. Daher müssen auch die Endgeschwindigkeiten den Startgeschwindigkeiten stark ähneln. In der untenstehenden Tabelle kann beobachtete werden, dass die Geschwindigkeiten zwar nicht ganz gleich sind, aber doch sehr ähnlich. (vgl. Tabelle 2)

Körper	Endgeschwindigkeit ($[\vec{v}] = {}^{m}/_{s}$)	Startgeschwindigkeit ($[\vec{v}] = {}^{m}/_{s}$)
Körper 1	$\vec{v} = \begin{pmatrix} 5.454271980219622 \cdot 10^2 \\ 1.0185132161977555 \cdot 10^3 \\ 0.0 \end{pmatrix}$	$\vec{v} = \begin{pmatrix} 5.7768070766 \cdot 10^2 \\ 1.00057233622 \cdot 10^3 \\ 0 \end{pmatrix}$
Körper 2	$\vec{v} = \begin{pmatrix} -1.1547719183284173 \cdot 10^3 \\ -3.690279869677892 \cdot 10^1 \\ 0.0 \end{pmatrix}$	$\vec{v} = \begin{pmatrix} -1.1553614153155713 \cdot 10^3 \\ 0 \\ 0 \end{pmatrix}$
Körper 3	$\vec{v} = \begin{pmatrix} 6.09344720306451 \cdot 10^2 \\ -9.816104175009901 \cdot 10^2 \\ 0.0 \end{pmatrix}$	$\vec{v} = \begin{pmatrix} 5.7768070766 \cdot 10^2 \\ -1.00057233622 \cdot 10^3 \\ 0 \end{pmatrix}$

Tabelle 2: Vergleich der berechneten Endgeschwindigkeiten mit den Startgeschwindigkeiten des Spezialfalls des Dreikörperproblems

Daraus kann abgeleitet werden, dass jeder der drei Körper einen oder mehrere vollständige Umläufe und zusätzlich ein kleines Stück absolviert hat.

6.1.1.4 Analyse der Endpositionen

Um die Endpositionen zu analysieren, wurden die Abstandsvektoren zwischen den Ortsvektoren der drei Körper berechnet, um die Seitenlänge des Dreiecks am Ende der Simulation zu erhalten. Die Abstände zwischen den Körpern am Ende sowie zu Beginn der Simulation sind in der untenstehenden Tabelle aufgelistet. (vgl. Tabelle 3)

Körper	Abstand zu Beginn $[s] = m$	Abstand am Ende $[s] = m$
Körper 1 und Körper 2	$s_{12} = 1 \cdot 10^{10}$	$s_{12} = 1.0000000000330769 \cdot 10^{10}$
Körper 2 und Körper 3	$s_{23} = 1 \cdot 10^{10}$	$s_{23} = 1.0000000000331709 \cdot 10^{10}$
Körper 1 und Körper 3	$s_{13} = 1 \cdot 10^{10}$	$s_{13} = 1.0000000000330246 \cdot 10^{10}$

Tabelle 3: Seitenlängen des gleichseitigen Dreiecks

Es ist erkennbar, dass die drei Körper am Ende der Simulation immer noch ein gleichseitiges Dreieck bilden. Bis auf kleine Fehler in der 12ten Nachkommastelle sind alle Seitenlänge gleich lang. Es ist erkennbar, dass die Seitenlängen des Dreiecks, welches die drei Körper bilden um gut $33\,cm$ angewachsen ist. Dies entspricht einer Abweichung von ca. $3.3 \cdot 10^{-9}$ %.

6.1.1.5 Analyse der Endgeschwindigkeiten

Um die Endgeschwindigkeiten zu analysieren, wird der Betrag der Geschwindigkeiten am Ende der Simulation mit dem Betrag der Geschwindigkeiten am Ende der Simulation verglichen. Dieser müsste gemäss der Theorie gleichbleiben, da sich die drei Körper mit konstanter Geschwindigkeit um ihren gemeinsamen Massenmittelpunkt bewegen. Die Anfangs- und Endgeschwindigkeiten sind für die drei Körper in der untenstehenden Tabelle aufgeführt. (vgl. Tabelle 4)

Körper	Startgeschwindigkeit $[v] = m/s$	Endgeschwindigkeit $[v] = m/s$
Körper 1	$v = 1.1553614153155713 \cdot 10^3$	$v = 1.1553614152773082 \cdot 10^3$
Körper 2	$v = 1.1553614153155713 \cdot 10^3$	$v = 1.1553614152772923 \cdot 10^3$
Körper 3	$v = 1.1553614153155713 \cdot 10^3$	$v = 1.155361415277408 \cdot 10^3$

Tabelle 4: Start- und Endgeschwindigkeiten der Körper im gleichseitigen Dreieck

Es wird sichtbar, dass auch die Geschwindigkeiten sehr konstant bleiben. Kleine Abweichungen treten ab der 10ten Nachkommastelle auf. Die drei Körper sind am Ende der Simulation etwas langsamer unterwegs als zu Beginn der Simulation. Dies passt mit der Abweichung der Seitenlängen zusammen, da Körper sich langsamer bewegen, wenn sie weiter vom Massenmittelpunkt entfernt sind, wie im 2. Keplerschen Gesetz beschrieben. Alle Körper bewegen sich jedoch auch am Ende der Simulation noch mit fast derselben Geschwindigkeit. Die Abweichung der Endgeschwindigkeiten von den Anfangsgeschwindigkeiten ist gleich gross wie die Abweichung der Seitenlängen und beträgt ebenfalls $3.3 \cdot 10^{-9}$ %.

6.1.1.6 Umlaufzeit

Wie bereits in 6.1.1.2 beobachtet werden konnte, haben die drei Körper während der Simulation etwas mehr als einen Umlauf absolviert. Die Umlaufzeit T der Körper beträgt:

$$T = 2\pi \sqrt{\frac{s^3}{G(m_1 + m_2 + m_3)}} = 31397956.34838304\ s$$

Die Gesamtlaufdauer ist etwas länger als die Umlaufzeit der Körper, nämlich $31556925.99\ s$. Der überstrichene Winkel ($[\varphi] = rad$) wird berechnet durch:

$$\Delta\varphi = \omega \cdot \Delta t = \sqrt{\frac{G\ (m_1 + m_2 + m_3)}{s^3}} \Delta t = 6.314997433040007$$

Der Winkel zwischen den Ortsvektoren der Anfangs- und der Endpositionen der drei Körper müsste daher $\alpha = 0.03181212586042115$ betragen. In der untenstehenden Tabelle sind die berechneten Zwischenwinkel zwischen den Ortsvektoren der Anfangs- und der Endposition des jeweiligen Körpers aufgelistet. (vgl. Tabelle 5)

Körper	Winkel zwischen Start- und Endposition $[\varphi] = rad$
Körper 1	$\varphi = 0.031945915932421416$
Körper 2	$\varphi = 0.03194591593241099$
Körper 3	$\varphi = 0.03194591593255698$

Tabelle 5: Zwischenwinkel der Ortsvektoren der Körper

Es ist erkennbar, dass die berechneten Winkel sehr genau mit dem erwarteten Winkel übereinstimmen. Die Zwischenwinkel der drei Körper sind bis auf die 11te Nachkommastelle identisch. Allerdings sind alle Zwischenwinkel etwas grösser als die aus der Theorie erwarteten. Daher haben sich die Körper etwas weiter bewegt, als gemäss der Theorie zu erwarten gewesen wäre.

6.1.1.7 Energieerhaltung

Alle drei Körper in diesem Spezialfall des Dreikörperproblems haben die gleiche Masse und bewegen sich mit konstanter Geschwindigkeit in einer Kreisbahn um ihren Massenmittelpunkt. Die kinetische Energie ist dabei unveränderlich, weil sich die Körper mit gleichbleibender Geschwindigkeit um ihren Massenmittelpunkt bewegen. Die potentielle Energie bleibt ebenfalls konstant, weil sich die Körper auf einer perfekten Kreisbahn befinden und damit ihr Abstand zu ihrem Massenmittelpunkt und somit auch ihr Potential konstant bleibt. Aus diesen Gründen gibt es auch keine Schwankungen in der Gesamtenergie. (vgl. Diagramm 1)

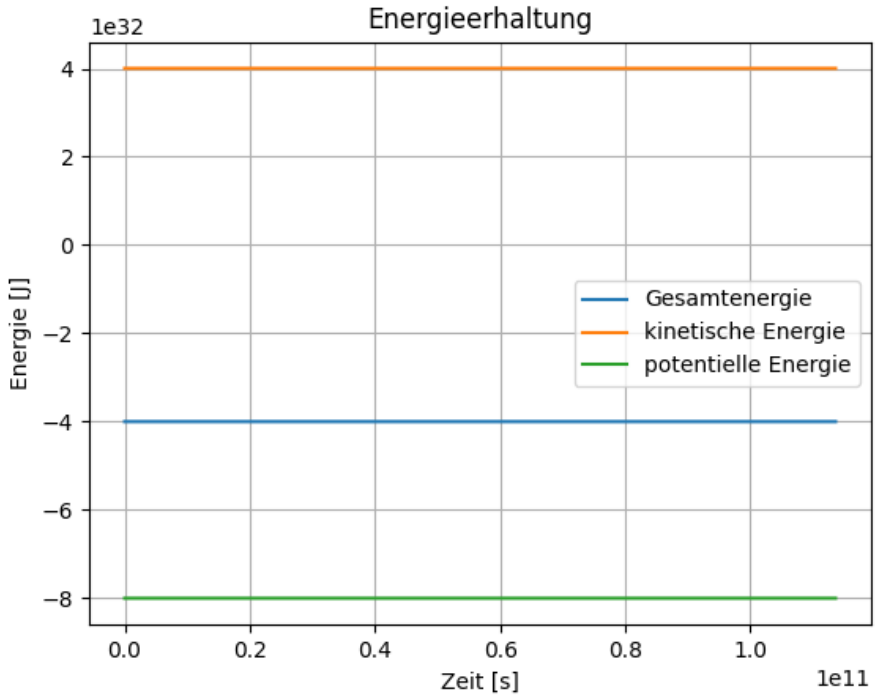


Diagramm 1: Energieerhaltung im Spezialfall des Dreikörperproblems: Gleichseitiges Dreieck

6.1.1.7.1 Schwankungen in den Energien

Dass die Gesamtenergie nicht schwankt, kann noch einmal im folgenden Diagramm nachvollzogen werden. Es wurde mit der Skalierung der y-Achse so hineingezoomt, dass die Schwankungen sichtbar werden. Es wird sichtbar, dass der Verlauf der Gesamtenergie zwar leicht gekrümmt ist, jedoch immer noch geradlinig verläuft. (vgl. Diagramm 2)

Die durchschnittliche Gesamtenergie beträgt ungefähr $-4.004579999999968 \cdot 10^{32}\,J$. Die Differenz zwischen dem grössten und dem kleinsten Wert der Gesamtenergie beträgt $1.0376293541461623 \cdot 10^{19}\,J$. Der grösste Wert der Gesamtenergie weicht damit um

$1.0976215324237847 \cdot 10^{-12}$ % vom Mittelwert ab und der kleinste Wert um $-1.4934850359208874 \cdot 10^{-12}$ %.

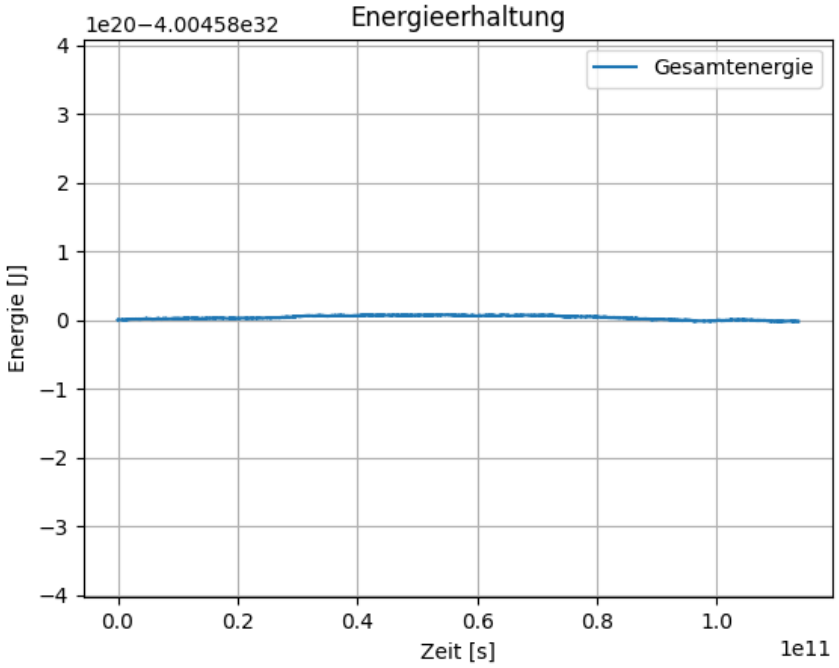


Diagramm 2: Schwankungen in der Gesamtenergie im Spezialfall des Dreikörperproblems: Gleichseitiges Dreieck

Stärker als die Gesamtenergie schwanken die kinetische und die potentielle Energien im System. Der Verlauf der kinetischen Energie gleicht dem Verlauf des Graphen der Funktion $f(x) = \cos(x)$, während der Verlauf der potentiellen Energie dem Verlauf des Graphen der Funktion $f(x) = -\cos(x)$ ähnelt. (vgl. Diagramm 3, Diagramm 4)

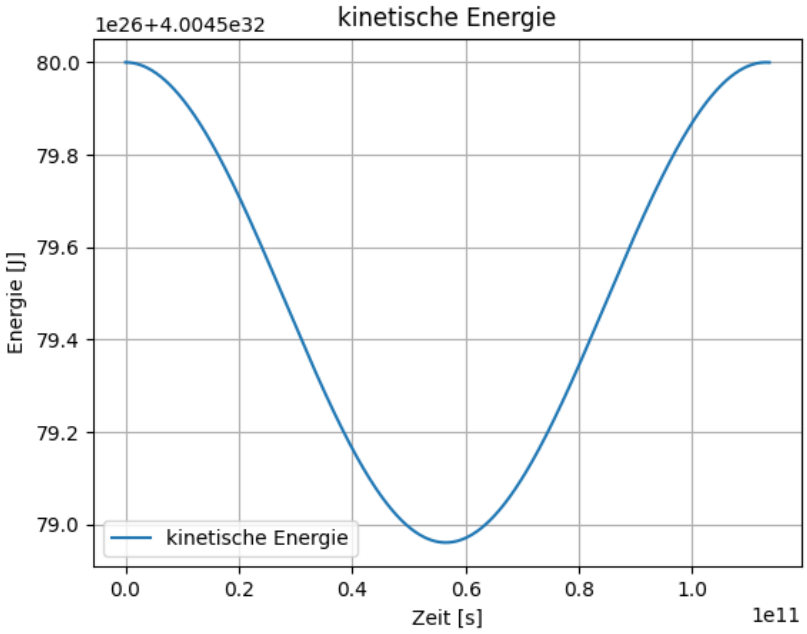


Diagramm 3: Schwankungen der kinetischen Energie im Spezialfall des Dreikörperproblems: Gleichseitiges Dreieck

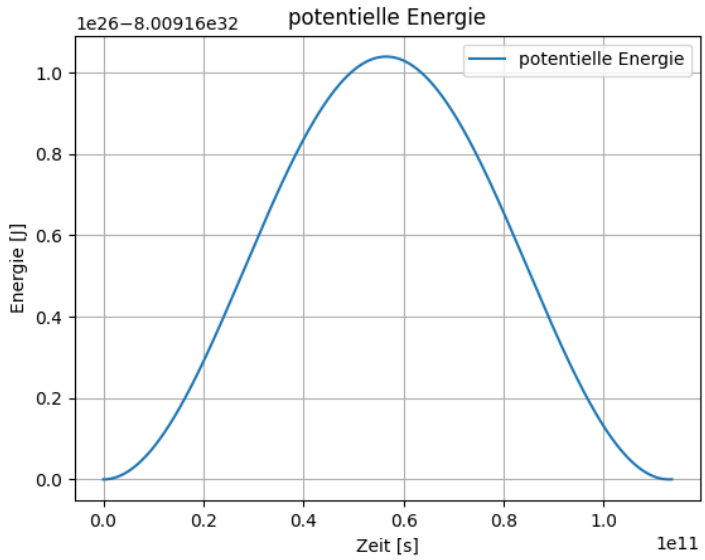


Diagramm 4: Schwankungen in der potentiellen Energie im Spezialfall des Dreikörperproblems: Gleichseitiges Dreieck

Aus den Schwankungen der potentiellen und der kinetischen Energie kann abgeleitet werden, dass die Bewegung der drei Körper in diesem Spezialfall des Dreikörperproblems nicht einer perfekten Kreisbahn entspricht und die Körper auch nicht zu jedem Zeitpunkt die genau gleiche Geschwindigkeit haben. In der Halbzeit der Simulation müssen sich alle Körper gleichermassen etwas vom Massenmittelpunkt entfernt haben, wodurch die Geschwindigkeiten der Körper und damit auch die gesamte kinetische Energie im System etwas gesunken sind und sich die potentielle Energie etwas vergrössert hat.

6.1.1.8 Impulserhaltung

Die Impulserhaltung in diesem Spezialfall des Dreikörperproblems ist gewährleistet. Der Gesamtimpuls bleibt sehr konstant. Dies wird auf dem untenstehenden Graphen verdeutlicht. (vgl. Diagramm 5)

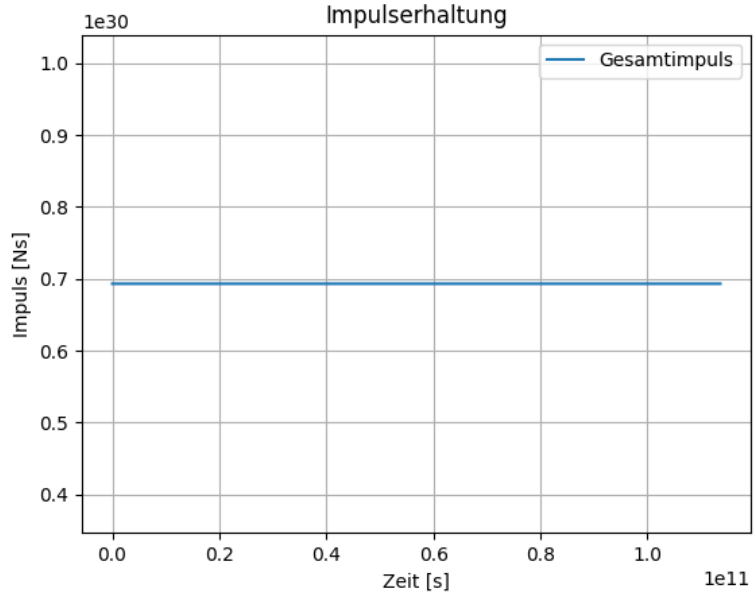


Diagramm 5: Impulserhaltung im Spezialfall des Dreikörperproblems: Gleichseitiges Dreieck

Auch wenn der Gesamtimpuls sehr konstant verläuft, gibt es trotzdem leichte Schwankungen im Gesamtimpuls. Diese werden im untenstehenden Graphen durch die Skalierung der y-Achse überdeutlich dargestellt. Es wird deutlich, dass der Gesamtimpuls wie die kinetische Energie im System einer Cosinus-Kurve über einer vollständigen Periode gleicht. Es kann vermutet werden, dass der Gesamtimpuls bei längerer Laufzeit des Programms oszilliert. Dass der Verlauf des Gesamtimpulses dem Verlauf der kinetischen Energie ähnelt, spricht für die Richtigkeit der Berechnungen, da die beiden Grössen von den gleichen Parametern abhängig sind. (vgl. Diagramm 6)

Der Mittelwert des Gesamtimpulses beträgt $6.932168044449328 \cdot 10^{29}\ Ns$. Die Differenz zwischen dem grössten und dem kleinsten Wert des Gesamtimpulses ist $8.994374677778352 \cdot 10^{22}\ Ns$. Der grösste Wert des Gesamtimpulses weicht damit um $6.454606173649806 \cdot 10^{-6}$ % vom Mittelwert ab und der kleinste Wert um $6.520230299018018 \cdot 10^{-6}$ %.

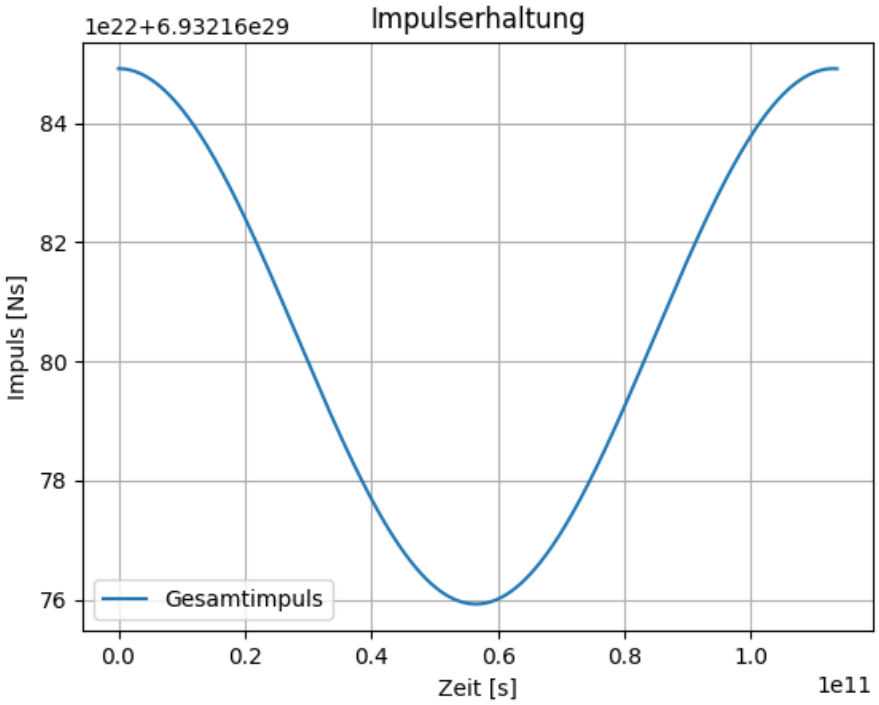


Diagramm 6: Schwankungen des Gesamtimpulses im Spezialfall des Dreikörperproblems: Gleichseitiges Dreieck

6.2 n-Körperproblem

6.2.1 Sonne und Planeten des Sonnensystems

Für die Analyse der Simulation zum n-Körperproblem wurden als Anfangsdaten die Ephemeriden der Himmelskörper verwendet. Diese wurden der *Horizons Web Application* der NASA entnommen. [26]

Die Ephemeriden (nur Position und Geschwindigkeit) wurden kopiert in ein File mit dem Namen *Himmelskörper_Daten* in den Dictionary unter den Eintrag mit dem Namen des Himmelskörpers hineinkopiert. Die Position wurde unter dem Key *'r'* und die Geschwindigkeit unter dem Key *'v'* als dreidimensionalen Array verpackt (vgl. 5.5.2). Der

Dictionary wurde in die Simulation des n-Körperproblems importiert. Die Simulation wurde über einen Zeitraum von 10 Jahren laufen gelassen. Zum Schluss der Simulation wurde ein statischer Plot ausgegeben, der die Bewegungen der verwendeten Himmelskörper aufzeigt und die Endpositionen nach der Berechnung sowie die Positionen der Körper gemäss den Ephemeriden des jeweiligen Körpers darüberlegt. (vgl., 5.2.2, 5.1)

6.2.1.1 Einstellungen der *Horizons Web Application*

Die erste Einstellung «Ephemeris Type» bestimmt, mit welchem Typ die Ephemeriden angegeben werden. Diese Einstellung wurde vom Default *Observer Table* zum *Vector Table* gewechselt.

Die zweite Einstellung «Target Body» definiert, für welchen Himmelskörper die Ephemeriden eingesehen werden sollen. Die untenstehende Tabelle zeigt, für welchen Körper welche Auswahl in der zweiten Einstellung getroffen wurde. (vgl. Tabelle 6)

Himmelskörper	**Name in den Einstellungen**
Sonne	Sun [Sol]
Merkur	Mercury
Venus	Venus
Erde	Earth
Mars	Mars
Jupiter	Jupiter
Saturn	Saturn
Uranus	Uranus
Neptun	Neptune

Tabelle 6: Namen der Himmelskörper in der Horizons Web Application

Die dritte Einstellung legt das Zentrum der Koordinaten fest. Dafür wurde unter *choose a method:* «search for a Location» angewählt. Es erscheint ein Fenster mit dem Titel «Lookup a Specified Location». Ins Suchfeld wurde *@sun* eingegeben. So wurde der Koordinatenursprung für die Ephemeriden auf die Sonne festgelegt.

Die vierte Einstellung heisst «Time Specifications». Sie legt fest, über welchem Zeitraum und in welchem zeitlichen Abstand die Ephemeriden angezeigt werden sollen. Dafür wurde als Startdatum (*Start time:*) der 23. September 2015 (*2015-09-23*) und als Enddatum (*End time:*) der 24. September 2015 (*2015-09-24*) eingefügt. Der zeitliche Abstand für die Ephemeriden wurde gemäss der Default Einstellung bei *1 days* belassen.

In der fünften Einstellung «Table Settings» wurden weitere Spezifikationen getätigt. Die Default Einstellungen sowie die angepassten Einstellungen werden in der untenstehenden Tabelle dargestellt. (vgl. Tabelle 7)

Name der Spezifikation	**Verwendete Spezifikation**	**Default**
Reference Frame:	ICRF	ICRF
Reference Plane	Ecliptic x-y plane derived from reference frame (standart obliquity, inertial)	Ecliptic x-y plane derived from reference frame (standart obliquity, inertial)
Vector correction:	geometric states	geometric states
Calendar type:	Gregorian	Mixed

Output units:	km and seconds	km and seconds
Vector labels:	□	✔
Output TDB-UT:	□	□
CSV format:	□	□
Object summary:	✔	✔

Tabelle 7: Spezifikationen der Einstellung 5 in der Horizons Web Application

6.2.1.2 Start-Ephemeriden für das n-Körperproblem

Mit den obigen Einstellungen für die *Horizons Web Application* wurden die Startephemeriden (23.09.2015 um 00:00 Uhr) für die in der Simulation verwendeten Körper entnommen. Dabei musste alle Werte mit dem Faktor 1000 multipliziert werden, da die Position in km und die Geschwindigkeit in $^{km}/_{s}$ angegeben wurden. In der untenstehenden Tabelle wird die Position in m und die Geschwindigkeit in $^{m}/_{s}$ angegeben. (vgl. Tabelle 8)

Körper	**Position ($[\vec{r}] = m$)**	**Geschwindigkeit ($[\vec{v}] = {}^{m}/_{s}$)**
Sonne	$\vec{r} = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}$	$\vec{v} = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}$
Merkur	$\vec{r} = \begin{pmatrix} 5.178819007555284 \cdot 10^{10} \\ -2.622832353990256 \cdot 10^{10} \\ -6.894403152705170 \cdot 10^{9} \end{pmatrix}$	$\vec{v} = \begin{pmatrix} 1.248058735416839 \cdot 10^{4} \\ 4.567438146577087 \cdot 10^{4} \\ 2.587026467585176 \cdot 10^{3} \end{pmatrix}$
Venus	$\vec{r} = \begin{pmatrix} 9.960859017735377 \cdot 10^{10} \\ 4.254457634969370 \cdot 10^{10} \\ -5.164750004286811 \cdot 10^{9} \end{pmatrix}$	$\vec{v} = \begin{pmatrix} -1.386975003246131 \cdot 10^{4} \\ 3.205282698244631 \cdot 10^{4} \\ 1.239812822175164 \cdot 10^{3} \end{pmatrix}$
Erde	$\vec{r} = \begin{pmatrix} 1.501334933289489 \cdot 10^{11} \\ -1.454019201043615 \cdot 10^{9} \\ -1.066517190239858 \cdot 10^{6} \end{pmatrix}$	$\vec{v} = \begin{pmatrix} -2.072238508139793 \cdot 10^{2} \\ 2.967115476465036 \cdot 10^{4} \\ -6.106368746241486 \cdot 10^{-1} \end{pmatrix}$
Mars	$\vec{r} = \begin{pmatrix} -1.591949732384840 \cdot 10^{11} \\ 1.882836748507884 \cdot 10^{11} \\ 7.852413482086882 \cdot 10^{9} \end{pmatrix}$	$\vec{v} = \begin{pmatrix} -1.758606716440902 \cdot 10^{4} \\ -1.358103815051285 \cdot 10^{4} \\ 1.470460221268102 \cdot 10^{2} \end{pmatrix}$
Jupiter	$\vec{r} = \begin{pmatrix} -7.326638935899717 \cdot 10^{11} \\ 3.380197182490783 \cdot 10^{11} \\ 1.499058577647360 \cdot 10^{10} \end{pmatrix}$	$\vec{v} = \begin{pmatrix} -5.633883245559018 \cdot 10^{3} \\ -1.125631273527197 \cdot 10^{4} \\ 1.728839579508903 \cdot 10^{2} \end{pmatrix}$
Saturn	$\vec{r} = \begin{pmatrix} -6.277204260422148 \cdot 10^{11} \\ -1.356561864659875 \cdot 10^{12} \\ 4.856955963160443 \cdot 10^{10} \end{pmatrix}$	$\vec{v} = \begin{pmatrix} 8.235404734168487 \cdot 10^{3} \\ -4.092627473967211 \cdot 10^{3} \\ -2.572985871137434 \cdot 10^{2} \end{pmatrix}$
Uranus	$\vec{r} = \begin{pmatrix} 2.841890862532807 \cdot 10^{12} \\ 9.274614570364714 \cdot 10^{11} \\ -3.335603862250805 \cdot 10^{10} \end{pmatrix}$	$\vec{v} = \begin{pmatrix} -2.165434721507191 \cdot 10^{3} \\ 6.145408130828825 \cdot 10^{3} \\ 5.132863545395461 \cdot 10^{1} \end{pmatrix}$
Neptun	$\vec{r} = \begin{pmatrix} 4.164599387125507 \cdot 10^{12} \\ -1.656186891604651 \cdot 10^{12} \\ -6.186269297035909 \cdot 10^{10} \end{pmatrix}$	$\vec{v} = \begin{pmatrix} 1.968709764849799 \cdot 10^{3} \\ 5.072291369137974 \cdot 10^{3} \\ -1.493328241014094 \cdot 10^{2} \end{pmatrix}$

Tabelle 8: Ephemeriden für das n-Körperproblem (Anfang)

6.2.1.3 End-Ephemeriden für das n-Körperproblem

Für die Kontrolle der Endpositionen der Körper in der Simulation zum n-Körperproblem wurden die Ephemeriden vom 23.09.2025 um 00:00 Uhr aller in der Simulation verwendeten Körper hinzugezogen, also die Position, die alle Körper nach exakt 10 Jahren haben. Dazu wurde für die *Horizons Web Application* die Einstellung 4 geändert. Das neue Startdatum ist der 23.09.2025 (*2025-09-23*) und das neue Enddatum ist der 24.09.2025 (*2025-09-24*). (vgl. 6.2.1.1)

Die aus der *Horizons Web Application* entnommenen Ephemeriden vom Enddatum sind ebenfalls in km und $^{km}/_{s}$ angegeben und müssen mit dem Faktor 1000 multipliziert werden, um die Position in m und die Geschwindigkeit in $^{m}/_{s}$ abzulesen, und so mit den berechneten Werten aus der Simulation verglichen werden können. Die Ephemeriden der beteiligten Körper sind in der untenstehenden Tabelle angegeben. (vgl. Tabelle 9)

Körper	**Position ($[\vec{r}] = \boldsymbol{m}$)**	**Geschwindigkeit ($[\vec{v}] = {}^{\boldsymbol{m}}/_{\boldsymbol{s}}$)**
Sonne	$\vec{r} = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}$	$\vec{v} = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}$
Merkur	$\vec{r} = \begin{pmatrix} -5.698223562961169 \cdot 10^{10} \\ -2.817902375598282 \cdot 10^{10} \\ 2.923546584427991 \cdot 10^{9} \end{pmatrix}$	$\vec{v} = \begin{pmatrix} 1.154513977882834 \cdot 10^{4} \\ -4.154624284941642 \cdot 10^{4} \\ -4.454172942644131 \cdot 10^{3} \end{pmatrix}$
Venus	$\vec{r} = \begin{pmatrix} -4.733700301766019 \cdot 10^{10} \\ 9.643132754115689 \cdot 10^{10} \\ 4.056141893573381 \cdot 10^{9} \end{pmatrix}$	$\vec{v} = \begin{pmatrix} -3.155654798118833 \cdot 10^{4} \\ -1.562789856945592 \cdot 10^{4} \\ 1.606110599824293 \cdot 10^{3} \end{pmatrix}$
Erde	$\vec{r} = \begin{pmatrix} 1.501301200761324 \cdot 10^{11} \\ -3.246005256339893 \cdot 10^{8} \\ -5.975128989698424 \cdot 10^{5} \end{pmatrix}$	$\vec{v} = \begin{pmatrix} -4.210615915036708 \cdot 10^{2} \\ 2.968838748326930 \cdot 10^{4} \\ -7.550404197367300 \cdot 10^{-1} \end{pmatrix}$
Mars	$\vec{r} = \begin{pmatrix} -1.521550253121659 \cdot 10^{11} \\ -1.755451126252503 \cdot 10^{11} \\ 5.245894385791570 \cdot 10^{7} \end{pmatrix}$	$\vec{v} = \begin{pmatrix} 1.922149510429560 \cdot 10^{4} \\ -1.379371338249704 \cdot 10^{4} \\ -7.604128451990189 \cdot 10^{2} \end{pmatrix}$
Jupiter	$\vec{r} = \begin{pmatrix} -1.429665740956006 \cdot 10^{11} \\ 7.609241834011847 \cdot 10^{11} \\ 3.781465404337645 \cdot 10^{7} \end{pmatrix}$	$\vec{v} = \begin{pmatrix} -1.300430792357781 \cdot 10^{4} \\ -1.802005992136028 \cdot 10^{3} \\ 2.984571094177807 \cdot 10^{2} \end{pmatrix}$
Saturn	$\vec{r} = \begin{pmatrix} 1.426771174142828 \cdot 10^{12} \\ -4.471440124888127 \cdot 10^{10} \\ -5.601498503516807 \cdot 10^{10} \end{pmatrix}$	$\vec{v} = \begin{pmatrix} -2.398563541165637 \cdot 10^{2} \\ 9.636071922321086 \cdot 10^{3} \\ -1.585135894894165 \cdot 10^{2} \end{pmatrix}$
Uranus	$\vec{r} = \begin{pmatrix} 1.529034569238336 \cdot 10^{12} \\ 2.485654879037999 \cdot 10^{12} \\ -1.059476126538372 \cdot 10^{10} \end{pmatrix}$	$\vec{v} = \begin{pmatrix} -5.863425547100653 \cdot 10^{3} \\ 3.252008748998560 \cdot 10^{3} \\ 8.783460694508327 \cdot 10^{1} \end{pmatrix}$
Neptun	$\vec{r} = \begin{pmatrix} 4.469783019533730 \cdot 10^{12} \\ 3.039168887282323 \cdot 10^{10} \\ -1.036283241921116 \cdot 10^{11} \end{pmatrix}$	$\vec{v} = \begin{pmatrix} -8.475475748400679 \cdot 10^{1} \\ 5.468146595298795 \cdot 10^{3} \\ -1.112104137028018 \cdot 10^{2} \end{pmatrix}$

Tabelle 9: Ephemeriden für das n-Körperproblem (Ende)

6.2.1.4 Vom Programm berechnete Endpositionen

Die Startephemeriden wurden als Startpositionen und Startgeschwindigkeiten für die Simulation des n-Körperproblems eingefügt. Danach wurden die Bewegungen der Planeten über die Zeitspanne von 10 Jahren mit dem Leapfrog-Algorithmus berechnet (vgl. 4.1, 5.1). Für die Berechnung der Anzahl Sekunden wurde die Definition des tropischen Jahres herangezogen, wobei ein Jahr 365.242199 Tage hat, weil nach einem tropischen Jahr die Erde sich wieder an der gleichen Position befindet. [27, p. 8]

Da alle Berechnungen in SI-Einheiten durchgeführt werden, muss diese Anzahl Tage noch mit dem Faktor $60\,{}^{s}/_{min} \cdot 60\,{}^{min}/_{h} \cdot 24\,{}^{h}/_{d} = 86400\,{}^{s}/_{d}$ multipliziert werden. Die Berechnung wurde demnach über $86400\,{}^{s}/_{d} \cdot 365.242199\,{}^{d}/_{y} \cdot 10y = 315569259.936\,s$ durchgeführt. Als Zeitschritt Δt wurden $3600s$ verwendet, da diese Grösse des Zeitschritts eine ausreichende Genauigkeit gewährleistet, und die Berechnungen in (einigermassen) absehbarer Zeit fertig werden.

Da die Simulation an der Position der Sonne fixiert wurde, bliebt diese stets im Ursprung des Koordinatensystems. Es ist daher keine Abweichung bei der Position der Sonne zu erwarten. Die Sonne hat jedoch am Ende der Simulation eine Geschwindigkeit, was sie gemäss den Ephemeriden nicht hat, um die Energie- und Impulserhaltung zu gewährleisten. Es ist erkennbar, dass, je weiter von der Sonne die Planeten sind, desto grösser sind auch die Komponenten ihrer berechneten Endpositionen. Im Gegensatz dazu nehmen die Komponenten der Endgeschwindigkeiten ab, je weiter aussen sich die Planeten im Sonnensystem befinden. Diese Entwicklungen der Positionen und Geschwindigkeiten war bereits bei den Anfangspositionen (durch die Ephemeriden gegeben) erkennbar und deutet auf die Richtigkeit der erhaltenen Resultate hin. (vgl. Tabelle 10, Tabelle 9)

Dadurch, dass die Sonne eine Endgeschwindigkeit hat, aber sie sich gemäss den Ephemeriden nicht bewegt, ist eine grössere Abweichung bei den Geschwindigkeiten der restlichen Körper zu erwarten.

Körper	**Position ($[\vec{r}] = \boldsymbol{m}$)**	**Geschwindigkeit ($[\vec{v}] = {}^{\boldsymbol{m}}/_{\boldsymbol{s}}$)**
Sonne	$\vec{r} = \begin{pmatrix} 0.0 \\ 0.0 \\ 0.0 \end{pmatrix}$	$\vec{v} = \begin{pmatrix} 9.641652159976902 \cdot 10^{0} \\ -1.2664490112184447 \cdot 10^{1} \\ -1.4940533242452173 \cdot 10^{-1} \end{pmatrix}$
Merkur	$\vec{r} = \begin{pmatrix} -5.750953886157459 \cdot 10^{10} \\ -2.6173070608786964 \cdot 10^{10} \\ 3.13584493835408 \cdot 10^{9} \end{pmatrix}$	$\vec{v} = \begin{pmatrix} 1.012926135739761 \cdot 10^{4} \\ -4.222943590273015 \cdot 10^{4} \\ -4.378368823256849 \cdot 10^{3} \end{pmatrix}$
Venus	$\vec{r} = \begin{pmatrix} -4.585856439450938 \cdot 10^{10} \\ 9.714998804608485 \cdot 10^{10} \\ 3.980628977934643 \cdot 10^{9} \end{pmatrix}$	$\vec{v} = \begin{pmatrix} -3.177926466970655 \cdot 10^{4} \\ -1.515763413377333 \cdot 10^{4} \\ 1.6260089858184137 \cdot 10^{3} \end{pmatrix}$
Erde	$\vec{r} = \begin{pmatrix} 1.4983323212455856 \cdot 10^{11} \\ -1.2140093031513536 \cdot 10^{10} \\ -1.1712827992286131 \cdot 10^{5} \end{pmatrix}$	$\vec{v} = \begin{pmatrix} 1.9306008150115736 \cdot 10^{3} \\ 2.9577635950135053 \cdot 10^{4} \\ -1.6018951675194797 \cdot 10^{0} \end{pmatrix}$
Mars	$\vec{r} = \begin{pmatrix} -1.5305006047335724 \cdot 10^{11} \\ -1.7489937286432455 \cdot 10^{11} \\ 8.795265619019282 \cdot 10^{7} \end{pmatrix}$	$\vec{v} = \begin{pmatrix} 1.9155849397622158 \cdot 10^{4} \\ -1.3892854371725396 \cdot 10^{4} \\ -7.605276235678186 \cdot 10^{2} \end{pmatrix}$
Jupiter	$\vec{r} = \begin{pmatrix} -1.439857157439016 \cdot 10^{11} \\ 7.606957387549867 \cdot 10^{11} \\ 5.79847195094719 \cdot 10^{7} \end{pmatrix}$	$\vec{v} = \begin{pmatrix} -1.2992344298312586 \cdot 10^{4} \\ -1.8295493588223726 \cdot 10^{3} \\ 2.9833913750662117 \cdot 10^{2} \end{pmatrix}$
Saturn	$\vec{r} = \begin{pmatrix} 1.4262549366395388 \cdot 10^{12} \\ -4.44920104372048 \cdot 10^{10} \\ -5.606634656900453 \cdot 10^{10} \end{pmatrix}$	$\vec{v} = \begin{pmatrix} -2.3200331514097684 \cdot 10^{2} \\ 9.62524745622038 \cdot 10^{3} \\ -1.5815849063879136 \cdot 10^{2} \end{pmatrix}$
Uranus	$\vec{r} = \begin{pmatrix} 1.5293202411140288 \cdot 10^{12} \\ 2.4855375477919453 \cdot 10^{12} \\ -1.049661565673487 \cdot 10^{10} \end{pmatrix}$	$\vec{v} = \begin{pmatrix} -5.853355233737717 \cdot 10^{3} \\ 3.2401824430259308 \cdot 10^{3} \\ 8.821878852441311 \cdot 10^{1} \end{pmatrix}$
Neptun	$\vec{r} = \begin{pmatrix} 4.469712426403482 \cdot 10^{12} \\ 3.0297454715887512 \cdot 10^{10} \\ -1.0340797453505875 \cdot 10^{11} \end{pmatrix}$	$\vec{v} = \begin{pmatrix} -7.482172648949155 \cdot 10^{1} \\ 5.456693664150079 \cdot 10^{3} \\ -1.1020573173195507 \cdot 10^{2} \end{pmatrix}$

Tabelle 10: Berechnete Endpositionen und Endgeschwindigkeiten

6.2.1.5 Darstellung der Simulation zum n-Körperproblem

Im untenstehenden Plot werden die berechneten Bahnen der Planeten dargestellt, wobei die berechnete Endposition als grosser Punkt in der entsprechenden Bahnfarbe dargestellt wurde und die Endposition gemäss den Ephemeriden als grüner Punkt darüber eingezeichnet wurde. (vgl. Abbildung 9)

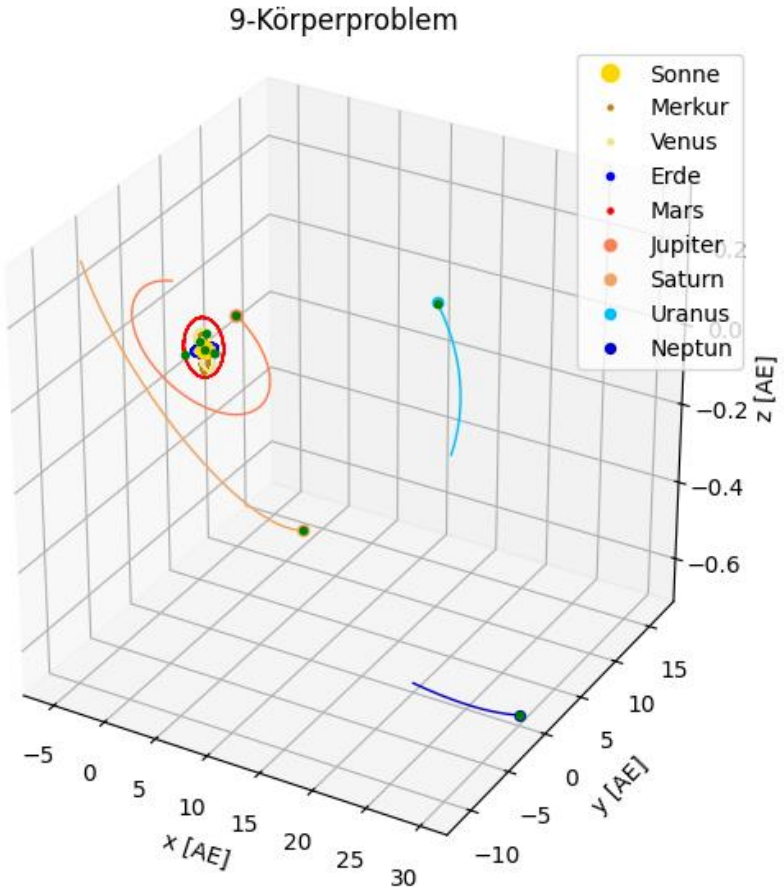


Abbildung 9: 9-Körperproblem mit den Körpern unseres Sonnensystems

Dabei kann festgestellt werden, dass die Positionen, welche durch die Ephemeriden gegeben sind, auf den berechneten Bahnen der Körper liegen und sich sehr in der Nähe der berechneten Positionen befinden. Allerdings sind die berechneten Positionen der äusseren Planeten deutlich genauer und weichen bei den inneren Planeten deutlich stärker ab. Ausserdem lässt sich feststellen, dass die Körper sich auf gekrümmten Bahnen bewegen und ihre Bahnen sich in fast der gleichen Ebene befinden.

Dass die Endpositionen der äusseren Planeten genauer sind als diejenigen der inneren Planeten, wird am untenstehenden Plot, welcher nur die inneren Planeten des Sonnensystems, d.h. die Bahnen von Merkur, Venus, Erde, Mars und einen Teil der Bahn von Jupiter, zeigt, besonders sichtbar. Vor allem die Endposition der Erde und die Endposition des Merkurs weisen Abweichungen von den erwarteten Endpositionen auf. Zudem sind alle berechneten Positionen den beobachteten Positionen etwas voraus.

Ausserdem ist erkennbar, dass bei den inneren Planeten, die die Sonne mehr als einmal umrundet haben, die Bahnen sehr genau aufeinander liegen. Auch dies lässt vermuten, dass die Berechnungen stimmen. (vgl. Abbildung 10)

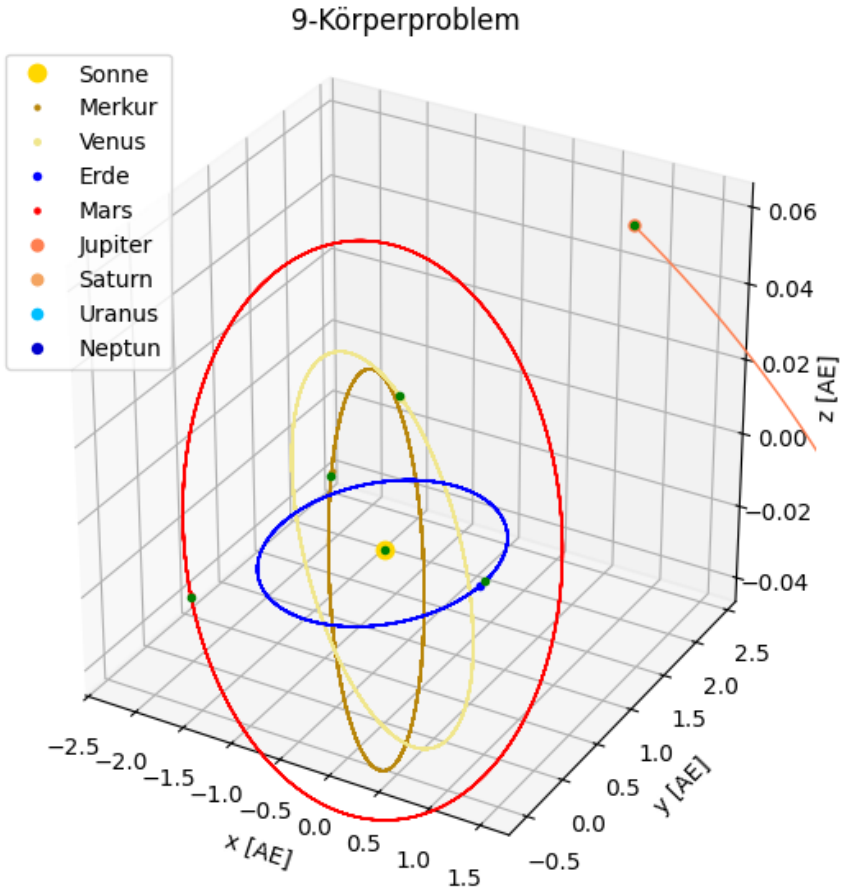


Abbildung 10: 9-Körperprolem mit den Körpern unseres Sonnensystems (innere Planeten)

6.2.1.6 Vergleich der berechneten Endpositionen mit den beobachteten Endpositionen

Bei der Betrachtung der berechneten Positionen mit den beobachteten Positionen fällt auf, dass die Vorzeichen (Plus oder Minus), die Potenzen und die ersten Stellen in den allermeisten Fällen stimmen, es aber bereits ab der ersten Nachkommastelle Abweichungen gibt. Besonders gross ist die Abweichung der y-Komponente und der z-Komponente der Position des Merkurs. Berechnet wurden für die y-Komponente $-2.6173070608786964 \cdot 10^{10}\ m$, gemäss den Ephemeriden müsste dieser Wert jedoch $-2.817902375598282 \cdot 10^{10}\ m$ lauten. Für die z-Komponente wurde der Wert $3.13584493835408 \cdot 10^{9}\ m$ berechnet. Die Ephemeriden bestimmen den Wert allerdings auf $2.923546584427991 \cdot 10^{9}\ m$. (vgl. Tabelle 13)

Ebenfalls eine grosse Diskrepanz wird bei der y-Komponente der Position der Erde sichtbar. Der berechnete Wert lautet $-1.2140093031513536 \cdot 10^{10}\ m$, während die Ephemeriden den Wert $-3.246005256339893 \cdot 10^{8}\ m$ geben.

Eine eher grössere Abweichung tritt auch bei der z-Komponente der Position des Mars auf. Der berechnete Wert dafür lautet $8.795265619019282 \cdot 10^{7}\ m$. Gemäss den Ephemeriden lautet die z-Komponente seiner Position aber $5.245894385791570 \cdot 10^{7}\ m$.

Die z-Komponente der berechneten Endposition des Jupiters weicht ebenfalls deutlich von derjenigen der beobachteten Position ab. Berechnet wurden dafür $5.79847195094719 \cdot 10^7\, m$, die Ephemeriden geben allerdings $3.781465404337645 \cdot 10^7\, m$ für die z-Komponente der Endposition des Jupiters.

Dadurch ist für die Positionen von Merkur, Erde, Mars und Jupiter eine höhere Abweichung zu erwarten. Die Positionen von Saturn und Uranus stimmen aussergewöhnlich gut mit den beobachteten Endpositionen überein. Dort ist demnach eine sehr kleine Abweichung zu erwarten. Da die Simulation wie auch die Ephemeriden die Position der Sonne als Koordinatenursprung festmacht, gibt es bei den Positionen der Sonne keine Abweichungen. (vgl. Tabelle 13)

Diese Erwartungen können mit der folgenden Tabelle bestätigt werden. Es wird sichtbar, dass die Abweichungen der berechneten Endpositionen bei den Planeten Merkur, Erde, Mars und Jupiter deutlich grösser sind als bei den restlichen Planeten. Ebenfalls erkennbar ist, dass die es bei den äussersten Planeten eine sehr geringe Abweichung bei den Positionen gibt. Besonders gross ist die Abweichung der Erde. Diese konnte bereits bei der Betrachtung der Abbildung 10 festgestellt werden. (vgl. Tabelle 11)

Körper	**Absolute Abweichung (als Vektor)** $([\vec{r}] = \boldsymbol{m})$	**Betrag der absoluten Abweichung** $([r] = \boldsymbol{m})$	**Relative Abweichung**
Sonne	$\vec{r} = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}$	$r = 0.0$	0.0 %
Merkur	$\vec{r} = \begin{pmatrix} 5.27303232 \cdot 10^8 \\ -2.00595315 \cdot 10^9 \\ -2.12298354 \cdot 10^8 \end{pmatrix}$	$r = 2.0849 \cdot 10^9$	10.2109 %
Venus	$\vec{r} = \begin{pmatrix} -1.47843862 \cdot 10^9 \\ -7.18660505 \cdot 10^8 \\ 7.55129156 \cdot 10^7 \end{pmatrix}$	$r = 1.6456 \cdot 10^9$	3.7116%
Erde	$\vec{r} = \begin{pmatrix} 2.96887952 \cdot 10^8 \\ 1.18154925 \cdot 10^{10} \\ -4.80384619 \cdot 10^5 \end{pmatrix}$	$r = 1.1819 \cdot 10^{10}$	3640.898%
Mars	$\vec{r} = \begin{pmatrix} 8.95035161 \cdot 10^8 \\ -6.45739761 \cdot 10^8 \\ -3.54937123 \cdot 10^7 \end{pmatrix}$	$r = 1.1042 \cdot 10^9$	67.6635 %
Jupiter	$\vec{r} = \begin{pmatrix} 1.01914165 \cdot 10^9 \\ 2.28444646 \cdot 10^8 \\ -2.01700655 \cdot 10^7 \end{pmatrix}$	$r = 1.0446 \cdot 10^9$	53.3441%
Saturn	$\vec{r} = \begin{pmatrix} 5.16237503 \cdot 10^8 \\ -2.22390812 \cdot 10^8 \\ 5.13615338 \cdot 10^7 \end{pmatrix}$	$r = 5.6444 \cdot 10^8$	0.5070%
Uranus	$\vec{r} = \begin{pmatrix} -2.85671876 \cdot 10^8 \\ 1.17331246 \cdot 10^8 \\ -9.81456086 \cdot 10^7 \end{pmatrix}$	$r = 3.2405 \cdot 10^8$	0.9266%
Neptun	$\vec{r} = \begin{pmatrix} 7.05931302 \cdot 10^7 \\ 9.42341569 \cdot 10^7 \\ -2.20349657 \cdot 10^8 \end{pmatrix}$	$r = 2.4983 \cdot 10^8$	0.3760%

Tabelle 11: Abweichungen der berechneten Endpositionen von den Positionen gemäss Ephemeriden

6.2.1.7 Vergleich der berechneten mit den beobachteten Endgeschwindigkeiten

Im Anhang befindet sich eine Tabelle, in der die berechneten Endgeschwindigkeiten und die von den Ephemeriden gegebenen Endgeschwindigkeiten als dreidimensionale Vektoren in $^{m}/_{s}$ angegeben sind. (vgl. Tabelle 14)

Da die Ephemeriden die Position der Sonne als Koordinatenursprung vorsehen, hat diese in keine Geschwindigkeit. Um die Energie- und Impulserhaltung jedoch zu gewährleisten, wird die Sonne während der Simulation beschleunigt und erhält eine Geschwindigkeit. Es ist erkennbar, dass die berechneten ziemlich gut mit den beobachteten Endgeschwindigkeiten übereinstimmen. Dies korreliert mit der guten Übereinstimmung der Positionen der Körper.

Bei fast allen berechneten Endgeschwindigkeiten stimmen die Potenzen, die ersten Stellen und die Vorzeichen der Komponenten Endgeschwindigkeiten. Bei der Erde tritt es in der x-Komponente der Endgeschwindigkeit eine grosse Abweichung auf. Sichtbar wird diese durch die Abweichung bereits in der ersten Stelle und des Vorzeichens. Die x-Komponente der Endgeschwindigkeit der Erde wurde auf $1.9306008150115736 \cdot 10^{3}\ ^{m}/_{s}$, die Ephemeriden geben allerdings einen Wert von $-4.210615915036708 \cdot 10^{2}\ ^{m}/_{s}$. Auch die z-Komponente der Endgeschwindigkeit der Erde weist eine Abweichung in der Potenz auf. Der berechnete Wert dafür lautet $-1.6018951675194797 \cdot 10^{0}\ ^{m}/_{s}$, während die Ephemeriden den Wert $-7.550404197367300 \cdot 10^{-1}\ ^{m}/_{s}$ geben. Auch die x-Komponente der Endgeschwindigkeit des Neptuns weist eine kleinere, aber sehr deutliche Abweichung aus. Berechnet wurden dafür $-7.482172648949155 \cdot 10^{1}\ ^{m}/_{s}$, die Ephemeriden sehen jedoch den Wert $-8.475475748400679 \cdot 10^{1}\ ^{m}/_{s}$ vor. Es kann dadurch erwartet werden, dass die Endgeschwindigkeit der Erde grosse Abweichungen, und die Endgeschwindigkeit des Neptuns verhältnismässig grösser Abweichungen aufweisen.

In der untenstehenden Tabelle sind die Abweichungen der Geschwindigkeiten der Sonne und der Planeten, als Vektor, als Betrag und als Prozentsatz eingetragen. Es wird ersichtlich, dass die berechnete Geschwindigkeit viel weniger von der beobachteten Geschwindigkeit abweicht als die Positionen voneinander abweichen. Bemerkenswert ist, dass die Geschwindigkeiten der inneren Positionen tendenziell stärker von den erwarteten Geschwindigkeiten abweichen als die Geschwindigkeiten der äusseren Planeten. Besonders gross ist die Abweichung der Endgeschwindigkeit der Erde. Dies korreliert mit der starken Abweichung der Endposition der Erde. Beim Planeten Uranus war zu sehen, dass die berechnete Endposition sehr gut mit der beobachteten Position übereinstimmt. Die gute Übereinstimmung der seiner Endgeschwindigkeit bestätigt dieses Ergebnis. (vgl. Tabelle 12)

Körper	**Absolute Abweichung (als Vektor) ($[\overrightarrow{\Delta v}] = {}^{m}/_{s}$)**	**Betrag der absoluten Abweichung ($[\Delta v] = {}^{m}/_{s}$)**	**Prozentuale Abweichung**
Sonne	$\overrightarrow{\Delta v} = \begin{pmatrix} 9.64165215998 \cdot 10^{0} \\ -1.26644901122 \cdot 10^{1} \\ -1.494053324245 \cdot 10^{-1} \end{pmatrix}$	$\Delta v = 15.9177$	
Merkur	$\overrightarrow{\Delta v} = \begin{pmatrix} 1.41587842143 \cdot 10^{3} \\ 6.8319305331 \cdot 10^{2} \\ -7.580411939 \cdot 10^{1} \end{pmatrix}$	$\Delta v = 1573.9157$	12.490%
Venus	$\overrightarrow{\Delta v} = \begin{pmatrix} 2.2271668852 \cdot 10^{2} \\ -4.7026443568 \cdot 10^{2} \\ -1.989838599 \cdot 10^{1} \end{pmatrix}$	$\Delta v = 520.7181$	3.330%

Erde	$\overrightarrow{\Delta v} = \begin{pmatrix} -2.35166241 \cdot 10^{3} \\ 1.10751533 \cdot 10^{1} \\ 8.46854748 \cdot 10^{-1} \end{pmatrix}$	$\Delta v = 2354.2690$	569.659%
Mars	$\overrightarrow{\Delta v} = \begin{pmatrix} 6.564570667 \cdot 10^{1} \\ 9.914098923 \cdot 10^{1} \\ 1.1477837 \cdot 10^{-1} \end{pmatrix}$	$\Delta v = 118.9046$	7.959%
Jupiter	$\overrightarrow{\Delta v} = \begin{pmatrix} -1.196362527 \cdot 10^{1} \\ 2.754336669 \cdot 10^{1} \\ 1.1797191 \cdot 10^{-1} \end{pmatrix}$	$\Delta v = 30.0296$	1.532%
Saturn	$\overrightarrow{\Delta v} = \begin{pmatrix} -7.85303898 \cdot 10^{0} \\ 1.08244661 \cdot 10^{1} \\ -3.8418158 \cdot 10^{-1} \end{pmatrix}$	$\Delta v = 13.3778$	3.284%
Uranus	$\overrightarrow{\Delta v} = \begin{pmatrix} -1.007031336 \cdot 10^{1} \\ 1.182630597 \cdot 10^{1} \\ -3.8418158 \cdot 10^{-1} \end{pmatrix}$	$\Delta v = 15.5377$	0.594%
Neptun	$\overrightarrow{\Delta v} = \begin{pmatrix} -9.93303099 \cdot 10^{0} \\ 1.145293115 \cdot 10^{1} \\ -1.00468197 \cdot 10^{0} \end{pmatrix}$	$\Delta v = 15.1936$	11.756%

Tabelle 12: Abweichungen der berechneten Endgeschwindigkeiten von Geschwindigkeiten gemäss Ephemeriden

6.2.1.8 Energieerhaltung

Die Gesamtenergie im System wird aus der potentiellen und der kinetischen Energie aller Körper zusammengesetzt. Die kinetische Energie wird grösser, wenn die potentielle Energie abnimmt. Dabei ist aussergewöhnlich, dass der Verlauf der kinetischen, wie auch der potentiellen Energie einer Cosinuskurve gleicht, welche über den Zeitraum der Berechnungen, also über 10 Jahre knapp keine vollständige Periode vollbracht hat. Die Gesamtenergie bleibt während der gesamten Laufzeit des Programms konstant bei ca. $-1.9666056 \cdot 10^{35}\ J$. Die Energieerhaltung ist somit gewährleistet. (vgl. Diagramm 7)

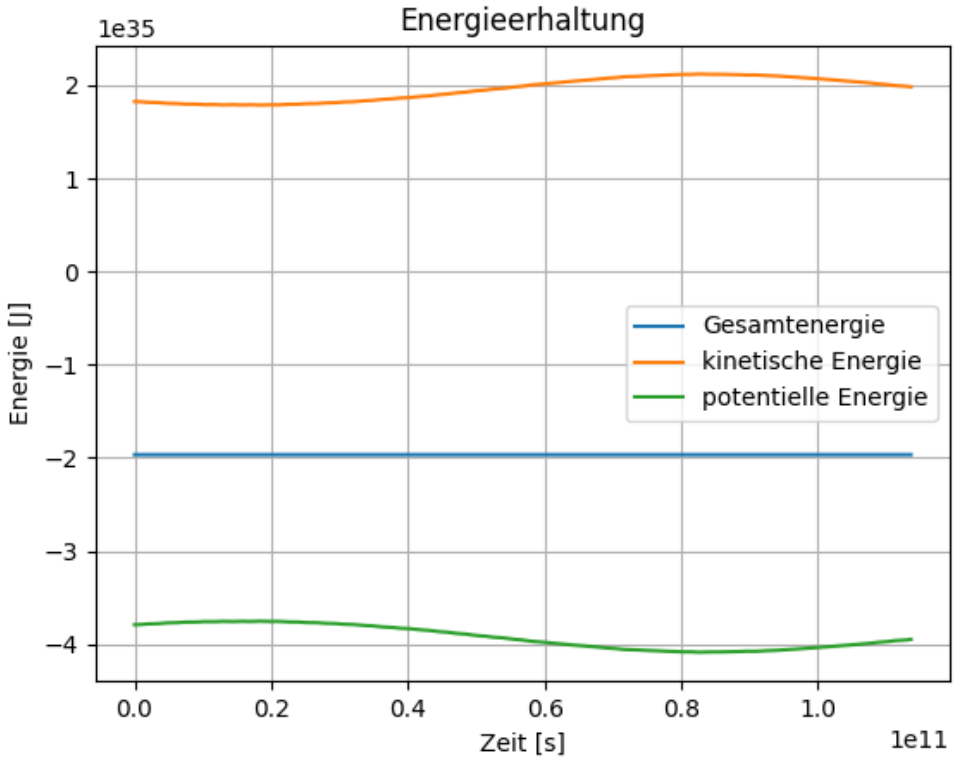


Diagramm 7: Energieerhaltung im n-Körperproblem

Es gibt jedoch trotzdem kleine oszillierende Veränderungen in der Gesamtenergie, welche beim Leapfrog-Algorithmus aber üblich sind (vgl. 4.1.1). Die Differenz zwischen dem grössten und dem kleinsten Wert der Gesamtenergie wird kleiner, je kleiner der Zeitschritt Δt gewählt wird. Die Gesamtenergie der Simulation des n-Körperproblems oszilliert über etwa $4.311181 \cdot 10^{26}\,J$. Damit weicht der kleinste Wert für die Gesamtenergie um etwa $1.30634562 \cdot 10^{-7}$ % und der grösste Wert um ca. $8.85848474 \cdot 10^{-9}$ % vom Mittelwert ab. Die Schwankungen in der Gesamtenergie sind also nicht beträchtlich. Bemerkenswert ist dazu, dass sich die Extrema der Gesamtenergie in zwei, relativ geraden und horizontalen Linien befinden. Es lässt dadurch darauf schliessen, dass die Schwankungen in der Gesamtenergie sehr gemässigt sind und es keine Ausreisser gibt. (vgl. Diagramm 8)

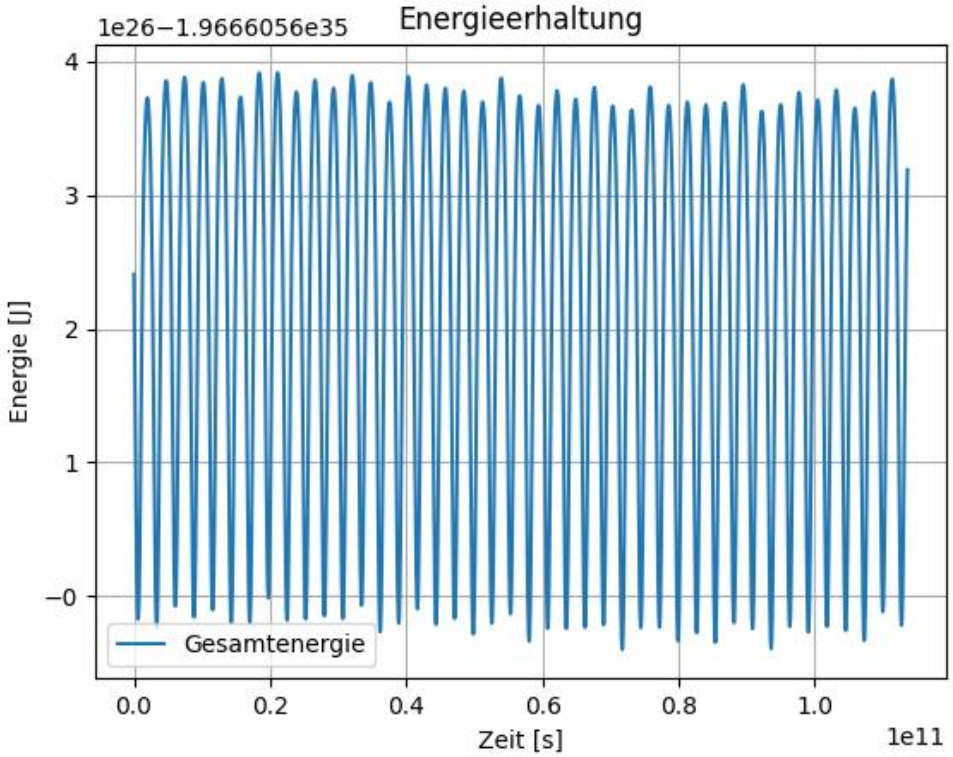


Diagramm 8: Oszillierende Gesamtenergie im n-Körperproblem

6.2.1.9 Impulserhaltung

Gemäss der Impulserhaltung muss auch der Impuls im ganzen System konstant bleiben. Dies ist hier der Fall. Der Graph des Gesamtimpulses bildet eine perfekte horizontale Linie. Man kann dadurch darauf schliessen, dass der Gesamtimpuls über den gesamten Zeitraum der Berechnungen absolut konstant bleibt. Der Wert des Gesamtimpulses im System liegt durchgehend bei ca. $2.3612092 \cdot 10^{31}\,J$. (vgl. Diagramm 9)

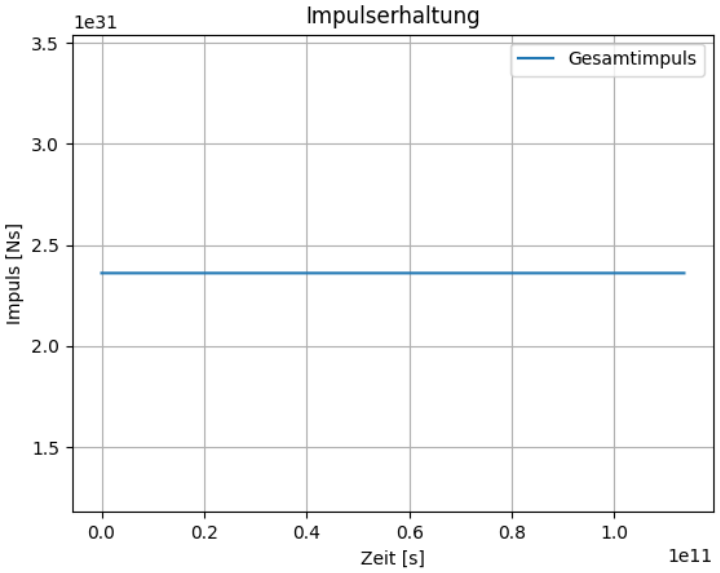


Diagramm 9: Impulserhaltung im n-Körperproblem

Das untenstehende Diagramm bestätigt diese Annahme. Bei Betrachtung eines Graphen, der die Schwankungen im Gesamtimpuls verdeutlicht, wird erkennbar, dass der Gesamtimpuls, im Gegensatz zur Gesamtenergie, nur minimalen Schwankungen unterlegen ist, und nicht wie die Gesamtenergie stark oszilliert. Der Graph sieht nicht aus wie derjenige einer Sinus- oder Cosinus-Funktion, sondern vielmehr einer nicht ganz perfekt gezogenen, aber trotzdem geraden und horizontalen Linie. (vgl. Diagramm 10)

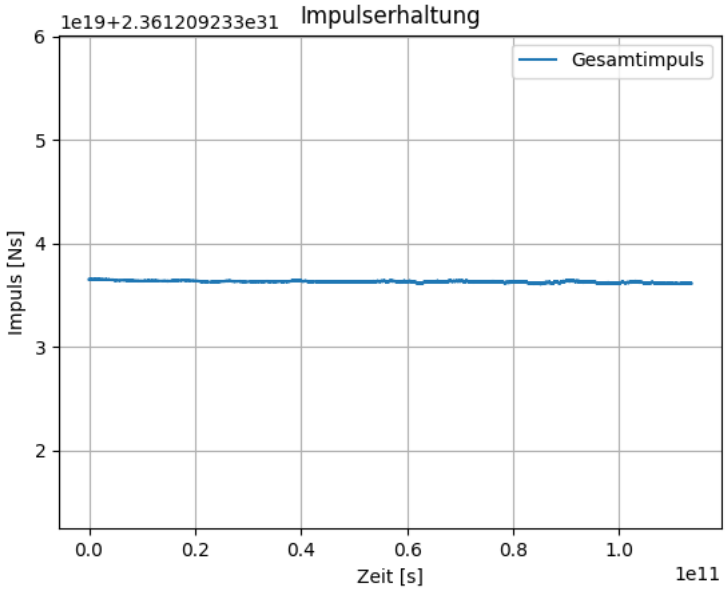


Diagramm 10: Schwankungen des Gesamtimpulses im n-Körperproblem

Die Differenz zwischen dem kleinsten und dem grössten Wert des Gesamtimpulses beträgt $6.800435437329449 \cdot 10^{17}\,Ns$. Der kleinste Wert weicht um etwa $-1.373275899 \cdot 10^{-12}\,\%$ und der grösste Wert um ca. $1.50678883 \cdot 10^{-12}\,\%$ vom Mittelwert des Gesamtimpulses ab.

7 Diskussion und Fazit

7.1 Simulation

Bei beiden analysierten Beispielen zur Simulation des Dreikörperproblems und n-Körperproblems konnte festgestellt werden, dass die berechneten Positionen sehr nahe an den erwarteten Positionen waren. Abgesehen von leichten Schwankungen wird mit der in dieser Arbeit behandelten Simulation auch die Energie- und Impulserhaltung gewährleistet.

7.1.1 Dreikörperproblem

In der Simulation zum Dreikörperproblem mit dem von Lagrange entdeckten Spezialfall des gleichseitigen Dreiecks wurde aufgrund der gleichbleibenden Geschwindigkeit und des stetigen Abstandes vom Massenmittelpunkt die Energieerhaltung hervorragend gewährleistet. Bei genauerer Betrachtung der kinetischen und der potentiellen Energie konnte festgestellt werden, dass die Körper sich in der Hälfte der Simulation etwas vom Mittelpunkt entfernten und dadurch etwas an Geschwindigkeit verloren haben. Diese Schwankungen sind allerdings so klein, dass sie in der Bewegung der Planeten nicht sichtbar werden.

Etwas weniger konstant als die Gesamtenergie war der Gesamtimpuls, der jedoch den gleichen Verlauf zeigte wie die kinetische Energie im System. Die Abweichungen der Extrema vom Mittelwert sind aber sehr klein. Ausserdem zeigt der Gesamtimpuls eine Tendenz zu oszillieren und wächst daher nicht stetig an oder fällt stetig ab. Deshalb kann festgestellt werden, dass in diesem Beispiel die Simulation sowohl die Energieerhaltung wie auch die Impulserhaltung gewährleistet.

7.1.2 n-Körperproblem

Bei der Beispielsimulation des n-Körperproblems wurden die Sonne und alle Planeten unseres Sonnensystems verwendet. Für die Simulation wurden Berechnungen über einen Zeitraum von 10 Jahren durchgeführt. Es konnte festgestellt werden, dass die Positionen der Körper gemäss den Ephemeriden nicht perfekt mit den berechneten Positionen übereinstimmten, jedoch auf den berechneten Bahnen lagen. Besonders grosse Abweichungen gab es bei den inneren Planeten. Die Abweichung der Endposition der Erde war besonders gross. In der Gesamtenergie wurden leichte Schwankungen festgestellt. Die Gesamtenergie oszillierte jedoch. Daher ist auch in diesem Beispiel der Simulation des n-Körperproblems die Energieerhaltung gewährleistet. Der Gesamtimpuls zeigt kaum Schwankungen und bleibt sehr konstant. Daher wird festgestellt, dass in diesem Beispiel der Simulation des n-Körperproblems die Impulserhaltung im System gewährleistet ist.

Bei der Analyse der berechneten Endpositionen und Endgeschwindigkeiten wurden leichte Abweichungen von den durch die Ephemeriden gegebenen Werten beobachtet. Diese Abweichung kann an der Berechnung der Gesamtlaufdauer des Programms liegen. Für die Berechnung der Anzahl Tage in den beobachteten 10 Jahren wurde die Länge des tropischen Jahrs verwendet. Die Berechnung ist zwar perfekt für die Berechnung der exakten Länge von 10 Jahren, unser Kalender funktioniert jedoch anders. Er schiebt Schalttage dazwischen, damit jedes Jahr eine ganzzahlige Anzahl von Tagen hat. Daher sind die 10 Jahre vom 23.09.2015 bis am 23.09.2025 etwas länger als die 10 Jahre vom 23.09.2017 bis am

23.09.2027 wären. Denn zwischen dem 23.09.2015 und dem 23.09.2025 wurden 3 Schalttage eingefügt, nämlich in den Jahren 2016, 2020 und 2024. Zwischen dem 23.09.2017 und dem 23.09.2027 wurden allerdings nur 2 Schalttage eingefügt, und zwar in den Jahren 2020 und 2024. Die Ephemeriden der *Horizons Web Application* der NASA richten sich jedoch nach dem Kalender und nicht nach der exakten Definition von Jahren. Daher ist der Zeitraum der Ephemeriden etwas kürzer als der Zeitraum, der für die Berechnungen definiert wurde. Aus diesem Grund sind auch alle berechneten Positionen der Körper den Ephemeriden etwas voraus, und umso mehr, je näher die Körper sich bei der Sonne befinden und je schneller sie sich daher bewegen.

Die Definition des tropischen Jahrs wurde trotzdem für die Simulation verwendet, da dieses die exakte Länge des Jahres darstellt und unempfindlich gegenüber der veränderlichen Anzahl von Tagen in einem Jahr oder in einer Zeitspanne von mehreren Jahren ist, und weil nach einem tropischen Jahr die Erde sich stets wieder an derselben Position befindet.

Da auch für Gravitationsprobleme Newton III gilt, wird auch die Sonne beschleunigt und bewegt sich. Alle Körper werden zwar nach jedem Zeitschritt so zurückverschoben, dass die Sonne sich wieder am Ursprung des Koordinatensystems befindet, sie erhält aber durch die Beschleunigung eine Geschwindigkeit, welche aufgrund der Energie- und Impulserhaltung nicht aufgehoben werden darf. Dadurch entstehen Abweichungen gegenüber den Ephemeriden, welche die Sonne als unbeweglichen Ursprung des Koordinatensystems heranziehen, wodurch sie gemäss den Ephemeriden keine Geschwindigkeit hat. Da die Energieerhaltung auch bei den Ephemeriden gilt, bewegt sich die Sonne in der Simulation schneller als sie sich gemäss den Ephemeriden und die Planeten bewegen sich langsamer als die Ephemeriden es vorgeben.

In der Simulation vom n-Körperproblem mit der Sonne und allen Planeten des Sonnensystems wurden nicht alle Körper im Sonnensystem berücksichtigt. Alle Monde, Zwergplaneten und der Asteroidengürtel wurden ausgelassen. Diese Körper wirken auch mit einer Gravitationskraft auf die anderen Körper und beeinflussen deren Bahn. Einen Teil der Abweichungen kann daher auch der Tatsache, dass die Körper isoliert betrachtet wurden, zugeschrieben werden. Die besonders grosse Abweichung der Planeten Erde, Mars und Jupiter kann dem Vernachlässigen des Asteroidengürtels zugeschrieben werden. Dieser wurde in der hier behandelten Simulation zum n-Körperproblem nicht beachtet, weil er aus zu vielen einzelnen Körpern besteht. Er hat aber durchaus einen Einfluss auf die näheren Planeten, da seine Gesamtmasse trotzdem relativ gross ist.

Da die Energie- und die Impulserhaltung immer gewährleistet sind und in der Simulation zu beobachten ist, dass sich die Planeten in regelmässigen, elliptischen Bahnen um die Sonne bewegen, liegt der Grund der Abweichung in der Zeitberechnung und nicht in der Richtigkeit der Simulation. Es kann daher festgehalten werden, dass diese Simulation, richtig angewandt, korrekte Ergebnisse ausgibt.

7.1.3 Generelle Ungenauigkeiten der Simulation

Python ist nicht der genaueste Taschenrechner. Für viele Berechnungen reicht er völlig aus, es kann aber ab der ca. 15ten Nachkommastelle zu kleineren Fehlern kommen. Normalerweise

haben diese Fehler keinen Einfluss auf das Ergebnis. Da bei der Simulation zum n-Körperproblem jedoch sehr viele Berechnungen durchgeführt werden, hat es einen Einfluss auf die Ungenauigkeiten in der Gesamtenergie und im Gesamtimpuls.

Der Leapfrog-Algorithmus, welcher für die Berechnungen der Simulation verwendet wurde, ist ein numerischer Integrationsalgorithmus. Der Zeitschritt Δt ist zwar klein, aber nicht 0. Es gibt Ungenauigkeiten, weil die Resultate des Leapfrog-Algorithmus nur Annäherungen an die wirklichen Werte darstellen, wenn die Annäherung beim Leapfrog-Algorithmus auch sehr gut ist.

7.1.4 Anwendungsbereich

Eine Simulation wie diejenige, welche in dieser Maturitätsarbeit erstellt wurde, werden unter anderem in der Raumfahrt verwendet, um beispielsweise Weltraumsonden und Satelliten auf stabile Bahnen zu führen, sodass sie nicht mit einem anderen Himmelskörper zusammenstossen. [30]

Für Körper, die uns (im astronomischen Sinne) nahe sind, und leicht mit dem Teleskop beobachtet werden können, werden die verschiedenen Punkte, an welchen sich dieser Körper befindet, gesammelt, und seine Umlaufbahn durch Interpolation berechnet. [31]

Für andere Körper, welche nicht leicht beobachtet werden können, weil sie entweder zu weit weg sind, zu klein sind oder zu wenig stark leuchten, könnte allerdings mithilfe einer Simulation, wie der hier behandelten, die Bahn und die aktuelle Position des Körpers berechnet werden, um anschliessend einen Anhaltpunkt für dessen Bahn zu erhalten. Ebenfalls könnte aufgrund der Vorhersage der Bewegung herausgefunden werden, dass es in der Nähe noch weitere massereiche Objekte gibt, wodurch sich der Körper nicht so bewegt, wie es durch die Simulation zu erwarten wäre.

Wenn die richtigen Anfangsbedingungen eingefügt werden und die Zeitspanne richtig berechnet wird, können mit der hier behandelten Position die Bewegungen von Himmelskörpern analysiert werden. Falls es Kollisionen gibt, kann diese Simulation diese allerdings nicht berechnen. Ausserdem braucht diese Simulation bei vielen Körpern und einer langen Laufdauer des Programms sehr lange, bis die Berechnungen abgeschlossen sind. Die animierte Simulation ist ebenfalls langsam und etwas stockend.

7.2 Arbeitsprozess

7.2.1 Beschreibung

7.2.1.1 Erste Schritte

Nach der Themenfindung widmete ich mich zuerst der Recherche über das Dreikörperproblem und suchte Literatur für die Berechnung der Bewegung von zunächst zwei Himmelskörpern (Zweikörperproblem). Dabei stiess ich auf Skripte, welche im ersten Jahr eines Physikstudiums verwendet werden und auf Formeln und Schreibweisen, welche ich nicht einmal teilweise verstand. Zeitgleich begann ich die ersten Kurzprogramme in Python zu schreiben, um mich mit der visuellen Datendarstellung etwas vertraut zu machen. Dabei halfen mir auch die Erkenntnisse, die ich aus dem Modul *PHY 124 Scientific Computing*, welches von Prof. Dr. Marcelle Soares dos Santos für Physik- und Mathematikstudenten im

ersten Semester angeboten wird und ich an der UZH als Schülerstudentin besuchen durfte, mitnehmen konnte.

Ich begann ein Konzept zu verfassen, erstellte einen Zeitplan, sammelte Informationen über die Ziele und Anforderungen einer Maturitätsarbeit und formulierte ein Minimalziel mit möglichen Erweiterungen. Anschliessend machte ich mich daran, meine Ziele zu erreichen. Ich versuchte ein Programm zum Zweikörperproblem zu schreiben. Als Anhaltspunkt für die Berechnungen versuchte ich aus Skripten von mehreren Universitäten die nötigen Formeln zu gewinnen. Da ich weder die Schreibweisen noch den Sinn der aufgeführten Formeln verstand, entstand auch kein brauchbares Programm. Ich begann neu auf einem weissen Blatt Papier, oder in diesem Fall in einem leeren Python-File. Dabei verwendete ich nur Informationen, die ich aus dem Physikunterricht an der Kantonsschule Zürcher Unterland erlangt hatte, vor allem aus der Himmelsmechanik und Information, welche ich aus dem besuchten UZH-Modul mitgenommen hatte. Dabei verwendete ich den Leapfrog-Algorithmus (inkorrekt, wie ich später feststellen musste), welchen ich während des Schüler*innenstudiums oft implementieren musste, und berechnete an jedem Punkt im Abstand von Δt die Gravitationskraft und die daraus resultierende Beschleunigung, ohne zu wissen, dass die Berechnung, die ich programmiert habe, numerisches Integrieren ist. Darauf hat mich mein Betreuer bei meinem nächsten Treffen hingewiesen und mir den Auftrag gegeben, zu verstehen, was die Integralrechnung ist und weshalb in meinem Programm zum Zweikörperproblem numerisch integriert wird. Hier steckte ich bis zum nächsten Treffen fest, weil ich das numerische Integrationsverfahren nicht vollumfänglich mit den Informationen über die Integralrechnung, welche ich auf Online-Lernplattformen fand, in Verbindung bringen konnte und ich nicht wusste, wie genau mein Verständnis über die Integralrechnung sein muss.

7.2.1.2 Programmierung des Dreikörperproblems

Bei meinem nächsten Treffen mit meinem Betreuer erhielt ich das Go, ein Programm zum Dreikörperproblem zu schreiben, so kam meine Maturitätsarbeit in vollen Gang. Ich begann zeitgleich die ersten Abschnitte im Theorieteil meiner Begleitarbeit zu schreiben und bemerkte während des Schreibens am Abschnitt über den Leapfrog-Algorithmus, welcher mich zwang, noch einmal über den Integrationsalgorithmus zu recherchieren, dass ich in meinem ursprünglichen Programm zum Zweikörperproblem diesen inkorrekt implementiert hatte. Daher musste ich das Δt für die Sicherstellung der Genauigkeit der Berechnungen relativ klein wählen, was das Programm sehr langsam machte. Ich erstellte ein Programm zum Dreikörperproblem, zunächst zweidimensional und statisch, mit korrekt implementiertem Leapfrog-Algorithmus, welches trotz eines zusätzlichen Himmelskörpers sehr viel schneller als das Programm zum Zweikörperproblem lief. Ausgehend von diesem ersten Programm zum Dreikörperproblem erstellte ich eine animierte Version in 2D, eine statische und eine animierte Version in drei Dimensionen. Ich versuchte anstatt des Leapfrog-Algorithmus die in der Python-Bibliothek vorhandene *scipy.integrate* numerische Integrationsfunktion *solve_ivp* zu verwenden und scheiterte dabei. Da die Laufzeit mit dem Leapfrog-Algorithmus immer noch überschaubar war, habe ich diese Option für die Berechnung des Dreikörperproblems nicht weiterverfolgt.

Während dieser gesamten Zeit wiesen meine Berechnungen zur Energie- und Impulserhaltung grosse Ungenauigkeiten auf. Die Diskrepanzen in der Impulserhaltung konnte ich erstmals mit dem Runden des Anfangs sowie des Endwerts beseitigen, aber die falschen Werte in der Energieerhaltung konnte ich dadurch nicht beheben.

7.2.1.3 n-Körperproblem

Ich fuhr weiter mit dem Programmieren und wagte mich ans n-Körperproblem (noch bevor ich den Fehler in der Energieerhaltung im Dreikörperproblem behoben hatte). Ich fand ein Skript, welches mir die Formeln für die Beschleunigung durch die Gravitation und für die gesamte kinetische und potentielle Energie im System gab. Ich verwendete viele *for*-Schleifen, um das Programm mit verschieden vielen Körpern laufen lassen zu können. Durch die neuen Formeln, die ich durch dieses Skript erlangt hatte, stimmte endlich die Energieberechnung, und dadurch auch die Gesamtenergie. Ich beschloss dann, die Werte der Gesamtenergie und des Gesamtimpulses nicht zu runden, da dies die erhaltenen Werte verfälscht. Auf der Suche nach den richtigen Geschwindigkeiten und Positionen der Planeten des Sonnensystems stiess ich auf Ephemeriden, Tabellen von beobachteten Positionen und Geschwindigkeiten mit Datum und Uhrzeit. Die Ephemeriden entnahm ich der *Horizons Web Application* der NASA (vgl. 6.2.1.1), bei welcher ich die richtigen Einstellungen treffen musste, damit die Daten in einem geeigneten Format ausgegeben wurden. Diese kopierte ich dann in die Dictionaries für die Anfangsdaten der Planeten (vgl. 5.5.2).

7.2.1.4 Simulationen

Ich begab mich anschliessend wieder zurück zum Dreikörperproblem und liess das Programm zum n-Körperproblem mit drei Körpern, deren Anfangspositionen und Anfangsgeschwindigkeiten so gewählt wurden, dass sie den Spezialfall des Dreikörperproblems des gleichseitigen Dreiecks darstellten, laufen. Anschliessend wagte ich mich auch noch an die Lagrange-Punkte, einem weiteren Spezialfall des Dreikörperproblems. Zu diesem Spezialfall fand ich allerdings keine Anfangspositionen und Anfangsgeschwindigkeiten, durch welche dieser korrekt dargestellt werden konnte. Aufgrund des näher rückenden Abgabetermins für die Maturitätsarbeit wurde dieser Spezialfall nicht mehr stärker verfolgt. Stattdessen musste ich das, was ich bereits programmiert hatte, in der Begleitarbeit festhalten und begann den Theorieteil zu vervollständigen, sowie das n-Körperproblem mit den Körpern unsers Sonnensystems zu analysieren.

Auf Anweisung meines Betreuers wagte ich noch einen Versuch, die Berechnungen zum n-Körperproblem mit dem Integrationsalgorithmus *solve_ivp* anstellen zu können und suchte einen Python-Code zum Dreikörperproblem, welcher die Positionen mit *solve_ivp* berechnete [28]. Ich versuchte analog dazu den Integrationsalgorithmus in mein Programm zum n-Körperproblem zu implementieren. Das Programm gab aber nie etwas Brauchbares aus. Ich begrub die Idee und verbesserte das Programm zum n-Körperproblem. Unter anderem gab ich der Sonne eine fixierte Position im Koordinatensystem.

7.2.1.5 Schriftliche Arbeit

Der Grundstein, um die schriftliche Arbeit weiterzuschreiben war mit der erfolgreichen Programmierung der Simulation zum n-Körperproblem gelegt. Jedoch passte die ursprüngliche Struktur der Arbeit, welche ich von der WiT-Arbeit übernommen habe, nicht

mehr zu meiner Maturitätsarbeit. Ich wusste nicht, was genau ich in die Arbeit hineinschreiben sollte und wo ich es einfügen sollte. Ich suchte daher nach wissenschaftlichen Arbeiten mit einem ähnlichen Thema, um deren Struktur zu übernehmen. Ich stiess auf eine Bachelorarbeit der Technischen Universität Dresden, von deren Aufteilung ich mich für meine eigene Arbeit inspirieren liess. [29]

Ich drückte mich allerdings stets vor dem Schreiben und arbeitete lieber an meinem Code, dadurch kam es, dass ich den Grossteil meiner schriftlichen Arbeit erst in den Weihnachtsferien schrieb. Das liegt unter anderem auch daran, dass ich meinen Python-Code nicht beschreiben wollte, bevor ich zu 100% zufrieden damit war, denn Änderungen im Code müssten auch in der Begleitarbeit beschrieben werden. Wenn zu früh angefangen wird, den Code zu beschreiben, schreibt man die Beschreibungen zweimal. Als die schriftliche Arbeit endlich Form angenommen hat, wurde sie von mir und von meiner Familie korrekturgelesen, noch ganz vervollständigt, Fehler wurden behoben und zuletzt wurde die Begleitarbeit gedruckt.

7.2.2 Beurteilung

Ich konnte mich lange sehr gut an den zu Beginn erstellten Zeitplan halten und hatte schon früh eine erste, funktionierende Simulation. Ich begann jedoch etwas zu spät, wirklich effizient an der Begleitarbeit zu schreiben und konnte für das Schreiben der Maturarbeit den Zeitplan nicht mehr einhalten. Dies liegt zum einen daran, dass ich unterschätzt hatte, wie viel Zeit das Schreiben der Begleitarbeit erfordert, zum anderen aber auch, weil ich zu Beginn nicht wusste, wie viel in der Arbeit beschrieben werden muss. Ausserdem habe ich unterschätzt, wie viel im Privaten im November läuft, und wie wenig Zeit ich in diesem Monat für die Maturitätsarbeit aufbringen kann.

Es gelang mir schon sehr früh, eine anständige Simulation mit Leapfrog zu programmieren, ich habe mehrere Versuche unternommen, eine Simulation mit dem Integrationsalgorithmus *solve_ivp*, der in der Bibliothek *scipy.integrate* zu finden ist, zu erstellen, welcher schneller und genauer gewesen wäre als der Leapfrog-Algorithmus. Allerdings wusste ich zu Beginn nicht, wo ich anfangen sollte. Ich hatte keinen Plan, wie die Funktion aussehen soll, die *solve_ivp* als Parameter verlangt und wie ich dies für mehr als einen Körper gleichzeitig berechnen kann. In einem letzten Versuch nahm ich das Beispiel einer Simulation des Dreikörperproblems in Python, welche die Berechnungen mit *solve_ivp* durchführte, zur Hilfe. Aber auch damit erfolgte die Bewegung der Planeten nicht so, wie sie sollte. Zum Schluss musste ich, unter anderem auch aus Zeitgründen, akzeptieren, dass die Simulation dieser Maturitätsarbeit mit dem Leapfrog-Algorithmus und nicht mit *solve_ivp* durchgeführt werden wird. Ich hätte gerne den Spezialfall des Dreikörperproblems mit den Lagrange-Punkten simuliert, konnte dies aber leider aus Zeitgründen nicht im Rahmen meiner Maturitätsarbeit realisieren.

7.3 Ausblick

Die in dieser Maturitätsarbeit erstellte Simulation ist bereits präzise und berücksichtigt die Energie und Impulserhaltung sehr gut. Sie ist jedoch ziemlich langsam, besonders für das in 6.2.1 analysierte Beispiel der Simulation mit der Sonne und den Planeten unseres

Sonnensystems. Daher könnte die Simulation in eine effizientere Programmiersprache übersetzt werden, damit die Berechnungen schneller und effizienter ablaufen. Dies wurde in dieser Maturitätsarbeit aus mangelnder Expertise in einer anderen Programmiersprache und aus Zeitgründen nicht vorgenommen.

Die Simulation könnte für einen allgemeinen Fall des Dreikörperproblems oder des n-Körperproblems ausgeweitet werden, in welchem die Körper sich nicht regelmässig, sondern sehr chaotisch bewegen und zusammenstossen können. In diesem Fall müssten Berechnungen aufgestellt werden, welche den Zusammenstoss von zweien Himmelskörpern beschreibt, oder die Simulation müsste abgebrochen werden, sobald zwei Körper sich an derselben Position befinden und dadurch zusammenstossen.

Mit der unveränderten Simulation könnten noch viele weitere Situationen des n-Körperproblems simuliert und analysiert werden, um einen tieferen Einblick in die Leistungsfähigkeit und den Nutzen der Simulation zu erlangen, was aus zeitlichen Gründen in dieser Maturitätsarbeit nicht gelungen ist. Es könnten zum Beispiel die Monde der Planeten in der Simulation berücksichtigt werden. Zusätzlich könnte der Spezialfall des Dreikörperproblem der Lagrange-Punkte simuliert und analysiert werden. Ausserdem könnten Was-wäre-wenn-Situationen simuliert werden. Beispielsweise könnten weitere, fiktive Planeten ins System hinzugefügt werden, oder die Positionen und Geschwindigkeiten zweier Planeten könnten vertauscht werden.

Dazu könnte man ein Programm schreiben, das die Ephemeriden der verwendeten Körper aus dem txt-File direkt in Python hineinlädt, sodass nicht manuell jede Position und jede Geschwindigkeit kopiert werden muss. Ausserdem könnte das System auf die Beachtung des Drehimpulses ausgeweitet werden und die Simulation unter Beachtung der Drehimpulserhaltung ausgewertet werden.

8 Danksagung

Ein grosses Dankeschön geht an meine Betreuungsperson Herrn Abreu für seine hilfreichen Inputs und Denkanstösse, die mir eine Richtung für meine Maturitätsarbeit gaben.

Des Weiteren möchte ich mich bei meiner Familie bedanken: Bei meiner Mutter für ein stets offenes Ohr und für die «Jetzt hättest du Zeit, willst du nicht noch etwas für deine Maturarbeit machen?». Bei meinem Vater fürs Korrekturlesen, kontrollieren der Zahlen und bei meiner Schwester für die Anstösse, die mir halfen, die wissenschaftliche Schreibweise in den Skripten zu verstehen.

Bei allen die mich verständnislos anschauten, als ich ihnen erzählte, dass ich als Maturitätsarbeit eine Simulation zum Dreikörperproblem programmiere, sodass ich lernte, die Thematik für Aussenstehende verständlich zu erklären, und bei allen, bei denen kein Fragezeichen auf der Stirn geschrieben stand.

Zum Schluss möchte ich mich bei all denjenigen bedanken, die mir das Schüler*innenstudium möglich gemacht haben, wodurch ich die Fertigkeiten erlernt habe, welche ich für meine Maturitätsarbeit brauchen konnte und welches mich auf das Thema für meine Maturitätsarbeit brachte.

9 Eigenständigkeitserklärung

Ich, Michaela Külling, erkläre hiermit, dass ich die vorliegende Maturitätsarbeit eigenständig und ohne unerlaubte fremde Hilfe erstellt habe und dass alle Quellen, Hilfsmittel und Internetseiten wahrheitsgetreu verwendet wurden und belegt sind. Zudem habe ich für die Erstellung der Maturitätsarbeit die KI nicht in unerlaubter Weise verwendet.

Datum: Unterschrift:

_____ _____

Ich bin damit einverstanden, dass eine Kopie meiner Maturitätsarbeit bei einer Anfrage nach aussen zum Lesen abgegeben werden darf.

10 Anhang

10.1 Tabellen

Körper	Berechnete Position ($[\vec{r}] = m$)	Position gemäss Ephemeriden ($[\vec{r}] = m$)
Sonne	$\vec{r} = \begin{pmatrix} 0.0 \\ 0.0 \\ 0.0 \end{pmatrix}$	$\vec{r} = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}$
Merkur	$\vec{r} = \begin{pmatrix} -5.750953886157459 \cdot 10^{10} \\ -2.6173070608786964 \cdot 10^{10} \\ 3.13584493835408 \cdot 10^{9} \end{pmatrix}$	$\vec{r} = \begin{pmatrix} -5.698223562961169 \cdot 10^{10} \\ -2.817902375598282 \cdot 10^{10} \\ 2.923546584427991 \cdot 10^{9} \end{pmatrix}$
Venus	$\vec{r} = \begin{pmatrix} -4.585856439450938 \cdot 10^{10} \\ 9.714998804608485 \cdot 10^{10} \\ 3.980628977934643 \cdot 10^{9} \end{pmatrix}$	$\vec{r} = \begin{pmatrix} -4.733700301766019 \cdot 10^{10} \\ 9.643132754115689 \cdot 10^{10} \\ 4.056141893573381 \cdot 10^{9} \end{pmatrix}$
Erde	$\vec{r} = \begin{pmatrix} 1.4983323212455856 \cdot 10^{11} \\ -1.2140093031513536 \cdot 10^{10} \\ -1.1712827992286131 \cdot 10^{5} \end{pmatrix}$	$\vec{r} = \begin{pmatrix} 1.501301200761324 \cdot 10^{11} \\ -3.246005256339893 \cdot 10^{8} \\ -5.975128989698424 \cdot 10^{5} \end{pmatrix}$
Mars	$\vec{r} = \begin{pmatrix} -1.5305006047335724 \cdot 10^{11} \\ -1.7489937286432455 \cdot 10^{11} \\ 8.795265619019282 \cdot 10^{7} \end{pmatrix}$	$\vec{r} = \begin{pmatrix} -1.521550253121659 \cdot 10^{11} \\ -1.755451126252503 \cdot 10^{11} \\ 5.245894385791570 \cdot 10^{7} \end{pmatrix}$
Jupiter	$\vec{r} = \begin{pmatrix} -1.439857157439016 \cdot 10^{11} \\ 7.606957387549867 \cdot 10^{11} \\ 5.79847195094719 \cdot 10^{7} \end{pmatrix}$	$\vec{r} = \begin{pmatrix} -1.429665740956006 \cdot 10^{11} \\ 7.609241834011847 \cdot 10^{11} \\ 3.781465404337645 \cdot 10^{7} \end{pmatrix}$
Saturn	$\vec{r} = \begin{pmatrix} 1.4262549366395388 \cdot 10^{12} \\ -4.44920104372048 \cdot 10^{10} \\ -5.606634656900453 \cdot 10^{10} \end{pmatrix}$	$\vec{r} = \begin{pmatrix} 1.426771174142828 \cdot 10^{12} \\ -4.471440124888127 \cdot 10^{10} \\ -5.601498503516807 \cdot 10^{10} \end{pmatrix}$
Uranus	$\vec{r} = \begin{pmatrix} 1.5293202411140288 \cdot 10^{12} \\ 2.4855375477919453 \cdot 10^{12} \\ -1.049661565673487 \cdot 10^{10} \end{pmatrix}$	$\vec{r} = \begin{pmatrix} 1.529034569238336 \cdot 10^{12} \\ 2.485654879037999 \cdot 10^{12} \\ -1.059476126538372 \cdot 10^{10} \end{pmatrix}$
Neptun	$\vec{r} = \begin{pmatrix} 4.469712426403482 \cdot 10^{12} \\ 3.0297454715887512 \cdot 10^{10} \\ -1.0340797453505875 \cdot 10^{11} \end{pmatrix}$	$\vec{r} = \begin{pmatrix} 4.469783019533730 \cdot 10^{12} \\ 3.039168887282323 \cdot 10^{10} \\ -1.036283241921116 \cdot 10^{11} \end{pmatrix}$

Tabelle 13: Vergleich der berechneten Endpositionen und den Endpositionen gemäss Ephemeriden

Körper	Berechnete Geschwindigkeit ($[\vec{v}] = {}^{m}/_{s}$)	Geschwindigkeit gemäss Ephemeriden ($[\vec{v}] = {}^{m}/_{s}$)
Sonne	$\vec{v} = \begin{pmatrix} 9.641652159976902 \cdot 10^{0} \\ -1.2664490112184447 \cdot 10^{1} \\ -1.4940533242452173 \cdot 10^{-1} \end{pmatrix}$	$\vec{v} = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}$
Merkur	$\vec{v} = \begin{pmatrix} 1.012926135739761 \cdot 10^{4} \\ -4.222943590273015 \cdot 10^{4} \\ -4.378368823256849 \cdot 10^{3} \end{pmatrix}$	$\vec{v} = \begin{pmatrix} 1.154513977882834 \cdot 10^{4} \\ -4.154624284941642 \cdot 10^{4} \\ -4.454172942644131 \cdot 10^{3} \end{pmatrix}$
Venus	$\vec{v} = \begin{pmatrix} -3.177926466970655 \cdot 10^{4} \\ -1.515763413377333 \cdot 10^{4} \\ 1.6260089858184137 \cdot 10^{3} \end{pmatrix}$	$\vec{v} = \begin{pmatrix} -3.155654798118833 \cdot 10^{4} \\ -1.562789856945592 \cdot 10^{4} \\ 1.606110599824293 \cdot 10^{3} \end{pmatrix}$
Erde	$\vec{v} = \begin{pmatrix} 1.9306008150115736 \cdot 10^{3} \\ 2.9577635950135053 \cdot 10^{4} \\ -1.6018951675194797 \cdot 10^{0} \end{pmatrix}$	$\vec{v} = \begin{pmatrix} -4.210615915036708 \cdot 10^{2} \\ 2.968838748326930 \cdot 10^{4} \\ -7.550404197367300 \cdot 10^{-1} \end{pmatrix}$
Mars	$\vec{v} = \begin{pmatrix} 1.9155849397622158 \cdot 10^{4} \\ -1.3892854371725396 \cdot 10^{4} \\ -7.605276235678186 \cdot 10^{2} \end{pmatrix}$	$\vec{v} = \begin{pmatrix} 1.922149510429560 \cdot 10^{4} \\ -1.379371338249704 \cdot 10^{4} \\ -7.604128451990189 \cdot 10^{2} \end{pmatrix}$
Jupiter	$\vec{v} = \begin{pmatrix} -1.2992344298312586 \cdot 10^{4} \\ -1.8295493588223726 \cdot 10^{3} \\ 2.9833913750662117 \cdot 10^{2} \end{pmatrix}$	$\vec{v} = \begin{pmatrix} -1.300430792357781 \cdot 10^{4} \\ -1.802005992136028 \cdot 10^{3} \\ 2.984571094177807 \cdot 10^{2} \end{pmatrix}$

Saturn	$\vec{v} = \begin{pmatrix} -2.3200331514097684 \cdot 10^2 \\ 9.62524745622038 \cdot 10^3 \\ -1.5815849063879136 \cdot 10^2 \end{pmatrix}$	$\vec{v} = \begin{pmatrix} -2.398563541165637 \cdot 10^2 \\ 9.636071922321086 \cdot 10^3 \\ -1.585135894894165 \cdot 10^2 \end{pmatrix}$
Uranus	$\vec{v} = \begin{pmatrix} -5.853355233737717 \cdot 10^3 \\ 3.2401824430259308 \cdot 10^3 \\ 8.821878852441311 \cdot 10^1 \end{pmatrix}$	$\vec{v} = \begin{pmatrix} -5.863425547100653 \cdot 10^3 \\ 3.252008748998560 \cdot 10^3 \\ 8.783460694508327 \cdot 10^1 \end{pmatrix}$
Neptun	$\vec{v} = \begin{pmatrix} -7.482172648949155 \cdot 10^1 \\ 5.456693664150079 \cdot 10^3 \\ -1.1020573173195507 \cdot 10^2 \end{pmatrix}$	$\vec{v} = \begin{pmatrix} -8.475475748400679 \cdot 10^1 \\ 5.468146595298795 \cdot 10^3 \\ -1.112104137028018 \cdot 10^2 \end{pmatrix}$

Tabelle 14: Vergleich der berechneten mit der beobachteten Geschwindigkeit

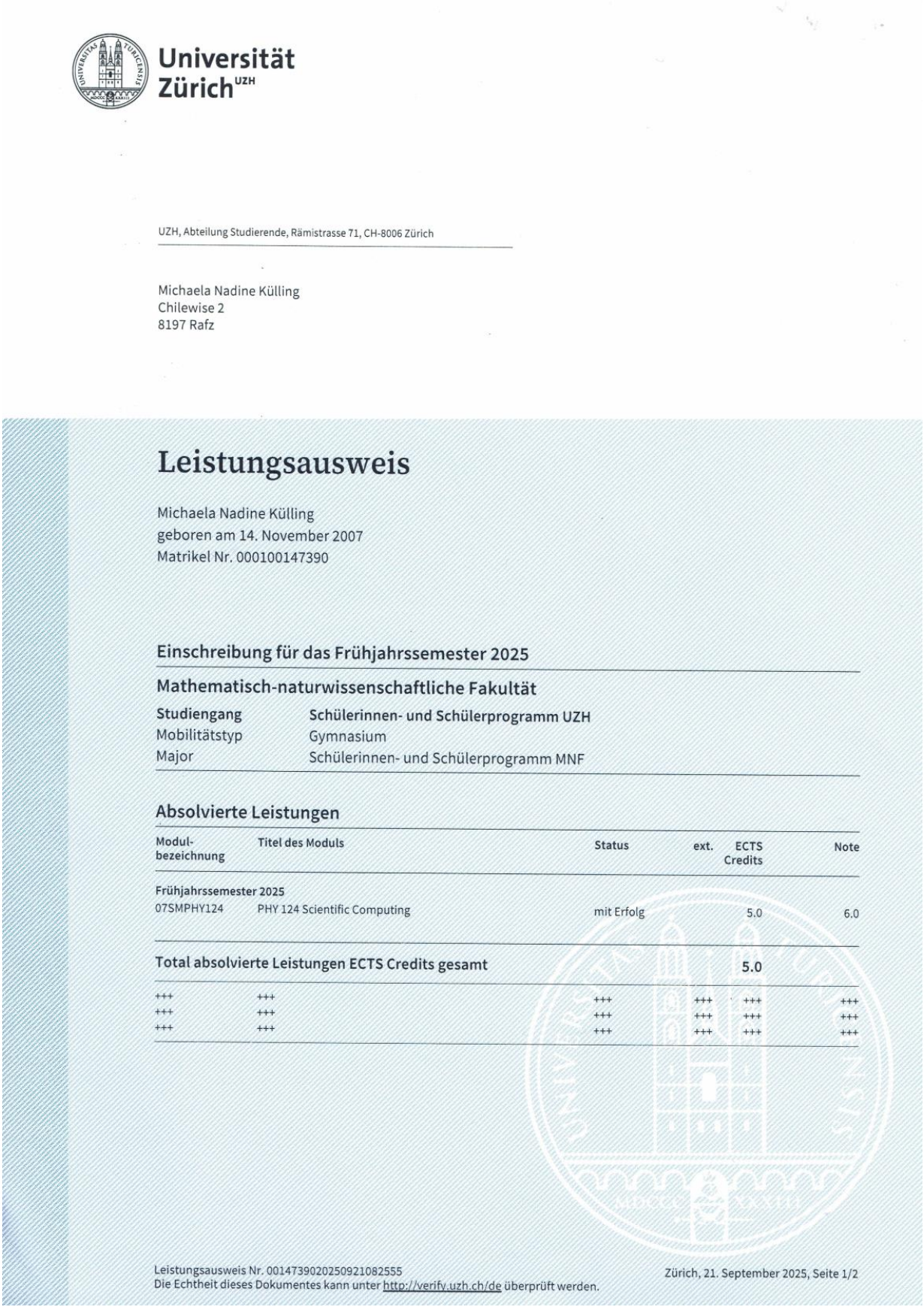
Körper	**Masse** $[\boldsymbol{m}] = \boldsymbol{kg}$
Sonne	$m = 1.9891 \cdot 10^{30}$
Merkur	$m = 3.30104 \cdot 10^{23}$
Venus	$m = 4.86732 \cdot 10^{24}$
Erde	$m = 5.97219 \cdot 10^{24}$
Mars	$m = 6.41693 \cdot 10^{23}$
Jupiter	$m = 1.89813 \cdot 10^{27}$
Saturn	$m = 5.68319 \cdot 10^{26}$
Uranus	$m = 8.68103 \cdot 10^{25}$
Neptun	$m = 1.02410 \cdot 10^{25}$

Tabelle 15: Massen der Planeten unseres Sonnensystems

Körper	**Farbe**	**Relative Grösse**
Sonne	'gold'	15
Merkur	'darkgoldenrod'	4
Venus	'khaki'	5
Erde	'blue'	6
Mars	‘red’	4.5
Jupiter	'coral'	10
Saturn	‘sandybrown’	9.5
Uranus	'deepskyblue'	9
Neptun	‘mediumblue’	8

Tabelle 16: Name, Farbe und Relative Grösse der Sonne und Planeten unseres Sonnensystems

10.2 Leistungsausweis UZH Modul PHY 124 Scientific Computing



Universität Zürich[UZH]

UZH, Abteilung Studierende, Rämistrasse 71, CH-8006 Zürich

Michaela Nadine Külling
Chilewise 2
8197 Rafz

Leistungsausweis

Michaela Nadine Külling
geboren am 14. November 2007
Matrikel Nr. 000100147390

Einschreibung für das Frühjahrssemester 2025

Mathematisch-naturwissenschaftliche Fakultät

Studiengang	**Schülerinnen- und Schülerprogramm UZH**
Mobilitätstyp	Gymnasium
Major	Schülerinnen- und Schülerprogramm MNF

Absolvierte Leistungen

Modul-bezeichnung	Titel des Moduls	Status	ext.	ECTS Credits	Note
Frühjahrssemester 2025					
07SMPHY124	PHY 124 Scientific Computing	mit Erfolg		5.0	6.0
Total absolvierte Leistungen ECTS Credits gesamt				**5.0**	
+++	+++	+++	+++	+++	+++
+++	+++	+++	+++	+++	+++
+++	+++	+++	+++	+++	+++

Leistungsausweis Nr. 001473902025092108255S
Die Echtheit dieses Dokumentes kann unter http://verify.uzh.ch/de überprüft werden.

Zürich, 21. September 2025, Seite 1/2

10.3 Python Files

n_Körperproblem_Berechnungen.py file:///C:/Users/bbg55/AppData/Roaming/Thonny/temp/thonny_t9f0u...

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.constants import au, G
import daten_speichern as ds

def n_Körperproblem(daten, n, Gesamtlaufdauer, delta_T, Fixierung, Vergleich, Zusatz):
    '''
    Input Parameter:
    daten:
        Dictionary mit Daten aller n Himmelskörper
        Für jeden Himmelskörper eigener Dictionary in daten mit Key '0', '1', ... str(n-1) mit:
            Masse (Key 'm')
            Anfangsposition (Key 'r')
            Anfangsgeschwindigkeit (Key 'v')
            Farbe des Körpers (Key 'farbe')
            Relative Grösse des Körpers (Key 'grösse')
            Name des Körpers (Key 'name')
            aus der Theorie erwartete Endposition nach Laufdauer des Programms (Key 'end_r')
wenn Vergleich == True
    n:
        Anzahl verwendeter Körper

    Gesamtlaufdauer:
        Gesamtzeit, über welche das Programm die Berechnungen durchführen soll in Sekunden

    delta_T:
        Zeitschritt für die Berechnung mit Leapfrog-Algorithmus in Sekunden

    Fixierung:
        Wenn True, wird die Position des Körpers mit Eintrag in daten['0'] am Ursprung fixiert

    Vergleich:
        Wenn True, werden im Plot die aus der Theorie erwarteten Endpositionen zusätzlich
        zu den berechneten Endpositionen geplottet

    Zusatz:
        Kürzel am Ende der Namen für die csv-Files als String

    Returns:
    csv-Files mit
        berechneten Positionen und Geschwindigkeiten der Körper,
        berechneten Werten der kinetischen und potentiellen Energie
        berechneten Werten der Gesamtenergie und der Gesamtimpulses

    Plot mit berechnetem n-Körperproblem, Plot zur Energieerhaltung, Plot zur Impulserhaltung
    '''


    def a_i(i):
        '''
        Input Parameter:
        i: Nummer des Körpers, für welchen die Beschleunigung berechnet werden soll

        Returns:
        berechnete Beschleunigung des Körpers i'''
        a_sum = 0
        for j in range(n):
            if i == j:
                continue
            else:
                a_sum += daten[str(j)]['m'] * ((daten[str(i)]['r_zwischen'] - daten[str(j)]
['r_zwischen'])/
                                               (np.linalg.norm(daten[str(i)]['r_zwischen'] -
daten[str(j)]['r_zwischen']))**3)
        a_sum *= -G
        return a_sum

```

```
65      def potentielle_Energie():
66          '''
67          Input Parameter:
68          None
69
70          Returns:
71          potentielle Energie im System'''
72
73          pot_sum = 0
74          for i in range(n):
75              for j in range(n):
76                  if i == j:
77                      continue
78                  else:
79                      pot_sum += (daten[str(i)]['m'] * daten[str(j)]['m'])/
(np.linalg.norm(daten[str(i)]['r_end'] - daten[str(j)]['r_end']))
80          pot_sum *= -(1/2)*G
81          return pot_sum
82
83      def kinetische_Energie():
84          '''Input Parameter:
85          None
86
87          Returns:
88          kinetische Energie im System'''
89
90          kin_sum = 0
91          for i in range(n):
92              kin_sum += (1/2) * daten[str(i)]['m'] * (np.linalg.norm(daten[str(i)]['v_new']))**2
93
94          return kin_sum
95
96      def Gesamtenergie ():
97          '''
98          Input Parameter:
99          None
100
101         Returns:
102         Gesamtenergie im System'''
103         return potentielle_Energie() + kinetische_Energie()
104
105     def Gesamtimpuls ():
106         '''
107         Input Parameter:
108         None
109
110         Returns:
111         Summe der Impulse der Körper 1, 2, 3 in N*s'''
112
113         def Impuls (m,v):
114             '''
115             Input Parameter:
116             m: Masse eines Körpers in kg
117             v: dreidimensionaler Array desselben Körpers als kartesische Koordinaten in m/s
118
119             Returns:
120             Betrag des berechneten Impulses des Körpers in N*s'''
121
122             return m * v
123
124         p_ges = 0
125         if n == 3:
126             for j2 in range (n):
127                 p_ges += np.linalg.norm(Impuls(daten[str(j2)]['m'], daten[str(j2)]['v_new']))
128         else:
129             for j3 in range (n):
130                 p_ges +=Impuls(daten[str(j3)]['m'], daten[str(j3)]['v_new'])
131
```

```
132         return np.linalg.norm(p_ges)
133
134
135
136
137     def leapfrog(T, dt):
138         '''
139         Input Parameter:
140         T: Zeitspanne für die Berechnung in s (Float oder Integer)
141         dt: Grösse der Zeitschritte in s (Integer)
142
143         Returns:
144         Tuple mit:
145             Liste mit den kinetischen Energien über die Zeitdauer der Berechnung
146             Liste mit den potentiellen Energien über die Zeitdauer der Berechnung
147             Liste mit den Gesamtenergien über die Zeitdauer der Berechnung
148             Liste mit den Gesamtimpulsen über die Zeitdauer der Berechnung
149
150         Einträge der Positionen und Geschwindigkeiten in den Dictionary der Körper
151         '''
152
153         # unter dem Key R findet sich eine Liste von Positionen der Körper 0, 1, ..., n
154         # Hier wird die Startposition der Körper 0, 1, ..., n eingefügt
155         for i1 in range(n):
156             daten[str(i1)]['R'] = [daten [str(i1)]['r'] ]
157
158         # unter dem Key V findet sich eine Liste von Geschwindigkeiten (als dreidimensionaler
    Array) der Körper 0, 1, ..., n
159         # Hier wird die Startgeschwindigkeit der Körper 0, 1, ..., n eingefügt
160         for i2 in range(n):
161             daten[str(i2)]['V'] = [daten [str(i2)]['v'] ]
162
163
164
165         # Energieberechnung
166         # Liste für potentielle Energie im System
167         E_pot = []
168         # Liste für kinetische Energie im System
169         E_kin = []
170         # Liste für Gesamtenergie
171         E_G = []
172
173         # Impulsberechnung
174         # Liste für Gesamtimpuls
175         p_G = []
176
177         for j in range (0, int(T), dt):
178             # Für alle Körper 0, 1, ..., n werden die Zwischenpositionen gemäss Leapfrog (r2 =
    r1 + v1 * dt/2)
179             for i3 in range(n):
180                 daten[str(i3)]['r_zwischen'] = daten[str(i3)]['R'][-1] + daten[str(i3)]['V'][-1]
    * dt/2
181
182             # Für alle Körper 0, 1, ..., n werden die Beschleunigungen durch die
    Gravitationskräfte berechnet (Funktion: a_i)
183             for i4 in range(n):
184                 daten[str(i4)]['a'] = a_i(i4)
185
186             # Für alle Körper 0, 1, ..., n werden die neuen Geschwindigkeiten berechnet gemäss
    Leapfrog (v2 = v1 + a * dt)
187             for i5 in range(n):
188                 daten[str(i5)]['v_new'] = daten[str(i5)]['V'][-1] + daten[str(i5)]['a'] * dt
189
190             # Für alle Körper 0, 1, ..., n werden die Endpositionen berechnet gemäss Leapfrog
    (r3 = r1 + v2 * dt/2)
191             for i6 in range(n):
192                 daten[str(i6)]['r_end'] = daten[str(i6)]['r_zwischen'] + daten[str(i6)]['v_new']
    * dt/2
```

```
193
194
195             # Falls Fixierung == True, Fixierung der Simulation an der Position des Körpers '0'
196             # -> Verschiebung aller Koordinaten, sodass sich die Körper '0' sich bei r = [0,0,0]
befindet
197             if Fixierung:
198                 delta_r = np.array([0,0,0]) - daten['0']['r_end']
199                 for i7 in range (n):
200                     daten[str(i7)]['r_end'] += delta_r
201
202
203             # Die Endpositionen und Endgeschwindigkeiten der Körper 0, 1, ..., n werden unter
dem Key R und V gespeichert
204             for i8 in range (n):
205                 daten[str(i8)]['R'].append(daten[str(i8)]['r_end'])
206                 daten[str(i8)]['V'].append(daten[str(i8)]['v_new'])
207
208
209             # Energieberechnung
210             E_pot.append (potentielle_Energie())
211             E_kin.append (kinetische_Energie())
212             E_G.append (Gesamtenergie())
213             # Impulsberechnung
214             p_G.append (Gesamtimpuls())
215
216         return E_pot, E_kin, E_G, p_G
217
218
219     E_pot, E_kin, E_G, p_G = leapfrog (Gesamtlaufdauer, delta_T)
220
221     # Energieerhaltung
222     print ('Energieerhaltung')
223     print ('Gesamtenergie (Ende), Gesamtenergie (Anfang)')
224     print (E_G[-1], E_G[0])
225     print ('ΔGesamtenergie')
226     print (E_G[-1] - E_G[0])
227     print ()
228
229     # Impulserhaltung
230     print ('Impulserhaltung')
231     print ('Gesamtimpuls (Ende), Gesamtimpuls (Anfang)')
232     print (p_G[-1], p_G[0])
233     print ('ΔGesamtimpuls')
234     print (p_G[-1] - p_G[0])
235
236     # Speicherung der Positionen und Geschwindigkeiten jeder Körper im jeweiligen csv-File
237     for i9 in range(n):
238         ds.save_csv_array(daten[str(i9)]['R'], str(i9)+'_Positionen' + Zusatz +'.csv')
239         ds.save_csv_array(daten[str(i9)]['V'], str(i9)+'_Geschwindigkeiten' + Zusatz +'.csv')
240     ds.save_csv_list(E_pot, 'potentielle_Energie' + Zusatz +'.csv')
241     ds.save_csv_list(E_kin, 'kinetische_Energie' + Zusatz +'.csv')
242     ds.save_csv_list(E_G, 'Gesamtenergie' + Zusatz +'.csv')
243     ds.save_csv_list(p_G, 'Gesamtimpuls' + Zusatz +'.csv')
244
245
246     # Erstellung der Listen für die Separierung der x-, y- und z-Koordinaten der Positionen der
Körper 0, 1, ..., n
247     for i10 in range (n):
248         daten[str(i10)]['x'] = []
249         daten[str(i10)]['y'] = []
250         daten[str(i10)]['z'] = []
251
252     # Separierung der x-, y- und z-Koordinaten der Positionen und gleichzeitige Umwandlung der
Koordinaten von m in AE
253     for j in range (0,len(daten['0']['R'])):
254         for i11 in range (n):
255             daten[str(i11)]['x'].append(daten[str(i11)]['R'][j][0]/au)
256             daten[str(i11)]['y'].append(daten[str(i11)]['R'][j][1]/au)
```

```
257             daten[str(i11)]['z'].append(daten[str(i11)]['R'][j][2]/au)
258
259
260
261     # Erstellen des Plots
262     fig = plt.figure()
263     ax = fig.add_subplot(1,3,1, projection="3d")
264     # Titel des Subplots
265     ax.set_title (str(n)+'-Körperproblem')
266     # Achsenbeschriftungen
267     ax.set_xlabel ('x [AE]')
268     ax.set_ylabel ('y [AE]')
269     ax.set_zlabel ('z [AE]')
270
271
272     # Darstellung der Bahnen der Körper
273     for i12 in range(n):
274         ax.plot(daten[str(i12)]['x'], daten[str(i12)]['y'], daten[str(i12)]['z'], color =
daten[str(i12)]['farbe'], linewidth = 0.5)
275
276     # berechnete Endposition
277     for i13 in range(n):
278         ax.plot(daten[str(i13)]['x'][-1], daten[str(i13)]['y'][-1], daten[str(i13)]['z'][-1],
linestyle = '', marker = '.', ms = daten[str(i13)]['grösse'], color = daten[str(i13)]['farbe'], label =
daten[str(i13)]['name'])
279
280     # erwartete Endposition
281     if Vergleich:
282         for i14 in range(n):
283             ax.plot(daten[str(i14)]['end_r'][0], daten[str(i14)]['end_r'][1], daten[str(i14)]
['end_r'][2], linestyle = '', marker = '.', color = 'green')
284
285     ax.legend()
286
287
288     x_E = np.linspace(0, int(Gesamtlaufdauer), len(E_pot))
289     x_E *= delta_T
290
291     ax = fig.add_subplot(1,3,2)
292     ax.set_title ('Energieerhaltung')
293     ax.set_xlabel ('Zeit [s]')
294     ax.set_ylabel ('Energie [J]')
295     ax.plot (x_E, E_G, label = 'Gesamtenergie')
296     ax.plot (x_E, E_kin, label = 'kinetische Energie')
297     ax.plot (x_E, E_pot, label = 'potentielle Energie')
298     ax.legend()
299
300     ax = fig.add_subplot (1,3,3)
301     ax.set_title ('Impulserhaltung')
302     ax.set_xlabel ('Zeit [s]')
303     ax.set_ylabel ('Impuls [Ns]')
304     ax.plot (x_E, p_G, label = 'Gesamtimpuls')
305     ax.set_ylim(0.5 * max(p_G) , 1.5*max(p_G))
306     ax.legend()
307
308
309
310
311     fig = plt.figure()
312     ax = fig.add_subplot(1,1,1, projection = '3d')
313     ax.set_title (str(n)+'-Körperproblem')
314     ax.set_xlabel ('x [AE]')
315     ax.set_ylabel ('y [AE]')
316     ax.set_zlabel ('z [AE]')
317     ax.set_aspect ('equal')
318
319     for i15 in range(n):
320         ax.plot(daten[str(i15)]['x'], daten[str(i15)]['y'], daten[str(i15)]['z'], color =
```

```
daten[str(i15)]['farbe'], linewidth = 1)
   321
   322      # berechnete Endposition
   323      for i16 in range(n):
   324          ax.plot(daten[str(i16)]['x'][-1], daten[str(i16)]['y'][-1], daten[str(i16)]['z'][-1],
linestyle = '', marker = '.', ms = daten[str(i16)]['grösse'], color = daten[str(i16)]['farbe'], label =
daten[str(i16)]['name'])
   325
   326      # erwartete Endposition
   327      if Vergleich:
   328          for i17 in range(n):
   329              ax.plot(daten[str(i17)]['end_r'][0], daten[str(i17)]['end_r'][1], daten[str(i17)]
['end_r'][2], linestyle = '', marker = '.', color = 'green')
   330
   331      ax.legend()
   332      plt.show()
   333
   334
```

```python
from n_Körperproblem_Berechnungen import n_Körperproblem
import Himmelskörper_Daten as hd


def Simulation_Lagrange_Dreieck ():
    '''
    Input Parameter
    None

    Returns:
    Simuliert den von Lagrange entdeckten Spezialfall des Dreikörperproblems
    mit drei Körpern der gleichen Masse, welche zusammen ein gleichseitiges
    Dreieck bilden und sich in einer Kreisbahn um ihren gemeinsamen
    Massenmittelpunkt bewegen.
    Dieses Beispiel ist fiktiv. Die Simulation spielt sich über den Zeitraum
    von einem Erdjahr ab.
    '''

    Daten = {}
    Daten['0'] = hd.data['Körper 1']
    Daten['1'] = hd.data['Körper 2']
    Daten['2'] = hd.data['Körper 3']

    # Gesamtzeit für Laufdauer des Programms
    Zeit = 3600 * 24 * 365.242199 * 1
    # Zeitschritt für Leapfrog-Algorithmus
    Zeitschritt = 3600

    # daten, n, Gesamtlaufdauer, delta_T, Fixierung, Vergleich, Zusatz
    n_Körperproblem(Daten, 3, Zeit, Zeitschritt, False, False, '_Lagrange_Dreieck')



def Simulation_Sonne_und_Planeten ():
    '''
    Input Parameter
    None

    Returns:
    Simuliert das n-Körperproblem mit der Sonne und allen Planeten unseres Sonnesystems
    Beginn der Simulation: 23.09.2015
    Ende der Simulation: 23.09.2025
    '''

    Daten = {}
    Daten['0'] = hd.data['Sonne']
    Daten['1'] = hd.data['Merkur']
    Daten['2'] = hd.data['Venus']
    Daten['3'] = hd.data['Erde']
    Daten['4'] = hd.data['Mars']
    Daten['5'] = hd.data['Jupiter']
    Daten['6'] = hd.data['Saturn']
    Daten['7'] = hd.data['Uranus']
    Daten['8'] = hd.data['Neptun']


    # Gesamtzeit für Laufdauer des Programms
    Zeit = 3600 * 24 * 365.242199 * 10

    # Zeitschritt für Leapfrog-Algorithmus
    Zeitschritt = 3600

    # daten, n, Gesamtlaufdauer, delta_T, Fixierung, Vergleich, Zusatz
    n_Körperproblem(Daten, 9, Zeit, Zeitschritt, True, True, '_Sonnensystem')


# Simuliert das n-Körperproblem mit der Sonne und allen Planeten unseres Sonnesystems
```

```
68 # Beginn der Simulation: 23.09.2015
69 # Ende der Simulation: 23.09.2025
70 Simulation_Sonne_und_Planeten ()
71
72
73 # Simuliert den Spezialfall des Dreikörperproblems: Gleichseitiges Dreieck
74 # Gesamtlaufdauer der Simulation: 1 Erdjahr
75 Simulation_Lagrange_Dreieck ()
76
77
```

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.constants import au
import csv
import Himmelskörper_Daten as hd

def Animation(daten, n, Zusatz, sl):
    '''
    Input Parameter:
    daten:
        Dictionary mit Daten aller n Himmelskörper
            Für jeden Himmelskörper eigener Dictionary in daten mit Key '0', '1', ... str(n-1) mit:
                Farbe des Körpers (Key 'farbe')
                Relative Grösse des Körpers (Key 'grösse')
                Name des Körpers (Key 'name')
    n:
        Anzahl verwendeter Körper
    Zusatz:
        Kürzel am Ende des Names des csv-Files
    sl:
        Länge der dargestellten, bereits absolvierten Bahn in Anzahl gespeicherter Positionen

    Returns:
    animierte Simulation'''

    for i1 in range (n):
        # öffnen des Files des Körpers mit dem Key str(i1)
        with open('D:/Files/' + str(i1) + '_Positionen'+ Zusatz +'.csv') as csvfile:
            daten[str(i1)]['reader'] = csv.reader(csvfile, delimiter=',', quotechar='"')

            # Erstellen der Listen für die x-, y- und z-Komponenten der Koordinaten der Positionen
            daten[str(i1)]['x'] = []
            daten[str(i1)]['y'] = []
            daten[str(i1)]['z'] = []

            # Einfügen der Komponenten der Koordinaten in die jeweiligen Listen und Umwandlung in AE
            for row in daten[str(i1)]['reader']:
                daten[str(i1)]['x'].append(float(row[0])/au)
                daten[str(i1)]['y'].append(float(row[1])/au)
                daten[str(i1)]['z'].append(float(row[2])/au)

    # Initiieren des Plots
    fig = plt.figure()
    ax = fig.add_subplot(projection="3d")

    # Berechnung der Grösse des Plots
    lim = max(daten[str(n-1)]['x'])

    for q in range (0, len(daten['0']['x']), 100):
        # Grösse des Plots festlegen
        ax.set_ylim (-lim, lim)
        ax.set_xlim (-lim, lim)
        ax.set_zlim (-lim, lim)

        # gleiche Skalierung der Achsen festlegen
        ax.set_aspect ('equal')

        # Achsenbeschriftungen
        ax.set_xlabel ('x [AE]')
        ax.set_ylabel ('y [AE]')
        ax.set_zlabel ('z [AE]')

        # Darstellung der bewegenden Punkte
        for i2 in range (n):
```

```
            ax.plot(daten[str(i2)]['x'][q], daten[str(i2)]['y'][q], daten[str(i2)]['z'][q],
marker = '.', color = daten[str(i2)]['farbe'], label = daten[str(i2)]['name'], ms = daten[str(i2)]
['grösse'])

        # Index für den letzten Punkt der darzustellenden Teilstrecke der bereits absolvierten
Bahn
        if q - sl < 0:
            p = 0
        else:
            p = q - sl

        # Darstellung der Teilstrecke der bereits absolvierten Bahn
        for i3 in range (n):
            ax.plot(daten[str(i3)]['x'][p:q], daten[str(i3)]['y'][p:q], daten[str(i3)]['z']
[p:q], color = daten[str(i3)]['farbe'])

        ax.legend()

        # Darstellung des Frames
        plt.pause (0.001)
        # Schliessen des Frames
        plt.cla()

def Animation_Lagrange_Dreieck():
    '''
    Input Parameter
    None

    Returns:
    Simuliert den von Lagrange entdeckten Spezialfall des Dreikörperproblems
    mit drei Körpern der gleichen Masse, welche zusammen ein gleichseitiges
    Dreieck bilden und sich in einer Kreisbahn um ihren gemeinsamen
    Massenmittelpunkt bewegen.
    Dieses Beispiel ist fiktiv. Die Simulation spielt sich über den Zeitraum
    von einem Erdjahr ab.
    '''

    Daten = {}
    Daten['0'] = hd.data['Körper 1']
    Daten['1'] = hd.data['Körper 2']
    Daten['2'] = hd.data['Körper 3']

    Animation(Daten, 3, '_Lagrange_Dreieck', 2900)

def Animation_Sonnensystem():
    '''
    Input Parameter
    None

    Returns:
    Simuliert das n-Körperproblem mit der Sonne und allen Planeten unseres Sonnesystems
    Beginn der Simulation: 23.09.2015
    Ende der Simulation: 23.09.2025
    '''

    Daten = {}
    Daten['0'] = hd.data['Sonne']
    Daten['1'] = hd.data['Merkur']
    Daten['2'] = hd.data['Venus']
    Daten['3'] = hd.data['Erde']
    Daten['4'] = hd.data['Mars']
    Daten['5'] = hd.data['Jupiter']
    Daten['6'] = hd.data['Saturn']
    Daten['7'] = hd.data['Uranus']
    Daten['8'] = hd.data['Neptun']

    Animation(Daten, 9, '_Sonnensystem', 10000)

```

```
129
130 Animation_Sonnensystem ()
131
132
133
```

```python
import csv

def save_csv_array(arr_lis, nam):
    '''
    input parameters:
    arr_lis: Liste mit Arrays, welche in einem csv file gespeichert werden soll
    nam: Name des csv-Files

    returns:
    speichert lis in csv file'''

    # neues File mit Namen 'nam' öffnen
    csvfile = open('D:/Files/' + nam, 'w', newline='', encoding='utf-8')
    c = csv.writer(csvfile)

    # Liste ins File einfügen
    c.writerows(arr_lis)

    # File schliessen und speichern
    csvfile.close()

def save_csv_list(lis, nam):
    '''
    input parameters:
    arr_lis: Liste mit floats, welche in einem csv file gespeichert werden soll
    nam: Name des csv files

    returns:
    speichert lis in csv file'''

    # neues File mit Namen 'nam' öffnen
    csvfile = open('D:/Files/' + nam, 'w', newline='', encoding='utf-8')
    c = csv.writer(csvfile)

    # Liste ins File einfügen
    for elem in lis:
        c.writerow([elem])

    # File schliessen und speichern
    csvfile.close()


```

```python
import numpy as np
from scipy.constants import au,G

data = {}

'''Daten für den Spezialfall des Dreikörperproblems: Gleichseitiges Dreieck'''
# Masse der Körper
m = 2e26
# Seitenlänge des Dreiecks, welches die drei Körper bilden
s = 1e10
# Inkreisradius
b = (np.sqrt(3)/6) * s
# Umkreisradius
r = (np.sqrt(3)/3) * s
# Halbe Seitenlänge
s_2 = s/2

# Betrag der Geschwindigkeit der drei Körper
v = np.sqrt(G*3*m/(s**3))*r

# cos(60°) -> für Aufteilung der Geschwindigkeit v in Geschwindigkeitsvektoren wichtig
cos60 = 1/2
# sin(60°) -> für Aufteilung der Geschwindigkeit v in Geschwindigkeitsvektoren wichtig
sin60 = np.sqrt(3)/2

# Eintrag für den Körper 1
data['Körper 1'] = {}
# Masse
data['Körper 1']['m'] = m
# Geschwindigkeit (in x,y,z Richtung (kartesische Koordinaten) als dreidimensionaler Array) in m/s
data['Körper 1']['v'] = np.array([cos60*v, sin60*v, 0])
# Position (in x,y,z Richtung (kartesische Koordinaten) als dreidimensionaler Array) in m
data['Körper 1']['r'] = np.array([s_2, -b, 0])
# Name des Himmelskörpers
data['Körper 1']['name'] = 'Körper 1'
# Farbe des Himmelskörpers
data['Körper 1']['farbe'] = 'red'
# Relative Grösse des Himmelskörpers
data['Körper 1']['grösse'] = 8


# Eintrag für den Körper 2
data['Körper 2'] = {}
# Masse
data['Körper 2']['m'] = m
# Geschwindigkeit (in x,y,z Richtung (kartesische Koordinaten) als dreidimensionaler Array) in m/s
data['Körper 2']['v'] = np.array([-v, 0, 0])
# Position (in x,y,z Richtung (kartesische Koordinaten) als dreidimensionaler Array) in m
data['Körper 2']['r'] = np.array([0, r, 0])
# Name des Himmelskörpers
data['Körper 2']['name'] = 'Körper 2'
# Farbe des Himmelskörpers
data['Körper 2']['farbe'] = 'pink'
# Relative Grösse des Himmelskörpers
data['Körper 2']['grösse'] = 8


# Eintrag für den Körper 3
data['Körper 3'] = {}
# Masse
data['Körper 3']['m'] = m
# Geschwindigkeit (in x,y,z Richtung (kartesische Koordinaten) als dreidimensionaler Array) in m/s
data['Körper 3']['v'] = np.array([cos60*v, -sin60*v, 0])
# Position (in x,y,z Richtung (kartesische Koordinaten) als dreidimensionaler Array) in m
```

```
65  data['Körper 3']['r'] = np.array([-s_2, -b, 0])
66  # Name des Himmelskörpers
67  data['Körper 3']['name'] = 'Körper 3'
68  # Farbe des Himmelskörpers
69  data['Körper 3']['farbe'] = 'violet'
70  # Relative Grösse des Himmelskörpers
71  data['Körper 3']['grösse'] = 8
72
73
74
75  '''Daten für das n-Körperproblem mit Sonne und allen Planeten unseres Sonnensystems'''
76  # Eintrag für die Sonne
77  data['Sonne'] = {}
78  # Masse
79  data['Sonne']['m'] = 1.988410e30
80  # Geschwindigkeit am 23.09.2015 um 00:00 (in x,y,z Richtung (kartesische Koordinaten) als
dreidimensionaler Array) in m/s
81  data['Sonne']['v'] = np.array([0,0,0])
82  # Position am 23.09.2015 um 00:00 (in x,y,z Richtung (kartesische Koordinaten) als
dreidimensionaler Array) in m
83  data['Sonne']['r'] = np.array([0,0,0])
84  # Name des Himmelskörpers
85  data['Sonne']['name'] = 'Sonne'
86  # Farbe des Himmelskörpers
87  data['Sonne']['farbe'] = 'gold'
88  # Relative Grösse des Himmelskörpers
89  data['Sonne']['grösse'] = 15
90  # Endposition gemäss Ephemeride vom 23.09.2025 um 00:00
91  data['Sonne']['end_r'] = np.array([0,0,0])
92
93
94  # Eintrag für den Merkur
95  data['Merkur'] = {}
96  # Masse
97  data['Merkur']['m'] = 3.302e23
98  # Geschwindigkeit am 23.09.2015 um 00:00 (in x,y,z Richtung (kartesische Koordinaten) als
dreidimensionaler Array) in m/s
99  data['Merkur']['v'] = np.array([1.248058735416839E+01,  4.567438146577087E+01,
2.587026467585176E+00]) *1000
100 # Position am 23.09.2015 um 00:00 (in x,y,z Richtung (kartesische Koordinaten) als
dreidimensionaler Array) in m
101 data['Merkur']['r'] = np.array([5.178819007555284E+07, -2.622832353990256E+07,
-6.894403152705170E+06]) *1000
102 # Name des Himmelskörpers
103 data['Merkur']['name'] = 'Merkur'
104 # Farbe des Himmelskörpers
105 data['Merkur']['farbe'] = 'darkgoldenrod'
106 # Relative Grösse des Himmelskörpers
107 data['Merkur']['grösse'] = 4
108 # Endposition gemäss Ephemeride vom 23.09.2025 um 00:00
109 data['Merkur']['end_r'] = np.array([-5.698223562961169E+07 *1000/au, -2.817902375598282E+07
*1000/au, 2.923546584427991E+06 *1000/au])
110
111
112 # Eintrag für die Venus
113 data['Venus'] = {}
114 # Masse
115 data['Venus']['m'] = 4.8685e24
116 # Geschwindigkeit am 23.09.2015 um 00:00 (in x,y,z Richtung (kartesische Koordinaten) als
dreidimensionaler Array) in m/s
117 data['Venus']['v'] = np.array([-1.386975003246131E+01,  3.205282698244631E+01,
1.239812822175164E+00]) *1000
118 # Position am 23.09.2015 um 00:00 (in x,y,z Richtung(kartesische Koordinaten) als
dreidimensionaler Array) in m
119 data['Venus']['r'] = np.array([9.960859017735377E+07,  4.254457634969370E+07,
-5.164750004286811E+06]) *1000
120 # Name des Himmelskörpers
121 data['Venus']['name'] = 'Venus'
```

```
122 # Farbe des Himmelskörpers
123 data['Venus']['farbe'] = 'khaki'
124 # Relative Grösse des Himmelskörpers
125 data['Venus']['grösse'] = 5
126 # Endposition gemäss Ephemeride vom 23.09.2025 um 00:00
127 data['Venus']['end_r'] = np.array([-4.733700301766019E+07 *1000/au,  9.643132754115689E+07
*1000/au, 4.056141893573381E+06 *1000/au])
128
129
130 # Eintrag für die Erde
131 data['Erde'] = {}
132 # Masse
133 data['Erde']['m'] = 5.97219e24
134 # Geschwindigkeit am 23.09.2015 um 00:00 (in x,y,z Richtung (kartesische Koordinaten) als
dreidimensionaler Array) in m/s
135 data['Erde']['v'] = np.array([-4.210615915036708E-01,  2.968838748326930E+01,
-7.550404197367300E-04]) * 1000
136 # Position am 23.09.2015 um 00:00 (in x,y,z Richtung (kartesische Koordinaten) als
dreidimensionaler Array) in m
137 data['Erde']['r'] = np.array([1.501301200761324E+08, -3.246005256339893E+05,
-5.975128989698424E+02]) * 1000
138 # Name des Himmelskörpers
139 data['Erde']['name'] = 'Erde'
140 # Farbe des Himmelskörpers
141 data['Erde']['farbe'] = 'blue'
142 # Relative Grösse des Himmelskörpers
143 data['Erde']['grösse'] = 6
144 # Endposition gemäss Ephemeride vom 23.09.2025 um 00:00
145 data['Erde']['end_r'] = np.array([1.501301200761324E+08 *1000/au, -3.246005256339893E+05 *1000/
au, -5.975128989698424E+02 *1000/au])
146
147
148 # Eintrag für den Mars
149 data['Mars'] = {}
150 # Masse in kg
151 data['Mars']['m'] = 6.4171e23
152 # Geschwindigkeit am 23.09.2015 um 00:00 (in x,y,z Richtung (kartesische Koordinaten) als
dreidimensionaler Array) in m/s
153 data['Mars']['v'] = np.array([-1.758606716440902E+01, -1.358103815051285E+01,
1.470460221268102E-01]) * 1000
154 # Position am 23.09.2015 um 00:00 (in x,y,z Richtung (kartesische Koordinaten) als
dreidimensionaler Array) in m
155 data['Mars']['r'] = np.array([-1.591949732384840E+08,  1.882836748507884E+08,
7.852413482086882E+06]) * 1000
156 # Name des Himmelskörpers
157 data['Mars']['name'] = 'Mars'
158 # Farbe des Himmelskörpers
159 data['Mars']['farbe'] = 'red'
160 # Relative Grösse des Himmelskörpers
161 data['Mars']['grösse'] = 4.5
162 # Endposition gemäss Ephemeride vom 23.09.2025 um 00:00 in AE
163 data['Mars']['end_r'] = np.array([-1.521550253121659E+08 *1000/au, -1.755451126252503E+08 *1000/
au, 5.245894385791570E+04 *1000/au])
164
165
166 # Eintrag für die Jupiter
167 data['Jupiter'] = {}
168 # Masse
169 data['Jupiter']['m'] = 1.89819e27
170 # Geschwindigkeit am 23.09.2015 um 00:00 (in x,y,z Richtung (kartesische Koordinaten) als
dreidimensionaler Array) in m/s
171 data['Jupiter']['v'] = np.array([-5.633883245559018E+00, -1.125631273527197E+01,
1.728839579508903E-01]) *1000
172 # Position am 23.09.2015 um 00:00 (in x,y,z Richtung (kartesische Koordinaten) als
dreidimensionaler Array) in m
173 data['Jupiter']['r'] = np.array([-7.326638935899717E+08,  3.380197182490783E+08,
1.499058577647360E+07]) *1000
174 # Name des Himmelskörpers
```

```
175 data['Jupiter']['name'] = 'Jupiter'
176 # Farbe des Himmelskörpers
177 data['Jupiter']['farbe'] = 'coral'
178 # Relative Grösse des Himmelskörpers
179 data['Jupiter']['grösse'] = 10
180 # Endposition gemäss Ephemeride vom 23.09.2025 um 00:00 in AE
181 data['Jupiter']['end_r'] = np.array([-1.429665740956006E+08 *1000/au, 7.609241834011847E+08
*1000/au, 3.781465404337645E+04 * 1000/au])
182
183
184 # Eintrag für den Saturn
185 data['Saturn'] = {}
186 # Masse
187 data['Saturn']['m'] = 5.6834e26
188 # Geschwindigkeit am 23.09.2015 um 00:00 (in x,y,z Richtung (kartesische Koordinaten) als
dreidimensionaler Array) in m/s
189 data['Saturn']['v'] = np.array([8.235404734168487E+00, -4.092627473967211E+00,
-2.572985871137434E-01]) *1000
190 # Position am 23.09.2015 um 00:00 (in x,y,z Richtung (kartesische Koordinaten) als
dreidimensionaler Array) in m
191 data['Saturn']['r'] = np.array([-6.277204260422148E+08, -1.356561864659875E+09,
4.856955963160443E+07]) *1000
192 # Name des Himmelskörpers
193 data['Saturn']['name'] = 'Saturn'
194 # Farbe des Himmelskörpers
195 data['Saturn']['farbe'] = 'sandybrown'
196 # Relative Grösse des Himmelskörpers
197 data['Saturn']['grösse'] = 9.5
198 # Endposition gemäss Ephemeride vom 23.09.2025 um 00:00 in AE
199 data['Saturn']['end_r'] = np.array([1.426771174142828E+09 *1000/au, -4.471440124888127E+07
*1000/au, -5.601498503516807E+07 *1000/au])
200
201
202 # Eintrag für den Uranus
203 data['Uranus'] = {}
204 # Masse
205 data['Uranus']['m'] = 8.6813e25
206 # Geschwindigkeit am 23.09.2015 um 00:00 (in x,y,z Richtung (kartesische Koordinaten) als
dreidimensionaler Array) in m/s
207 data['Uranus']['v'] = np.array([-2.165434721507191E+00,  6.145408130828825E+00,
5.132863545395461E-02]) *1000
208 # Position am 23.09.2015 um 00:00 (in x,y,z Richtung (kartesische Koordinaten) als
dreidimensionaler Array)  in m
209 data['Uranus']['r'] = np.array([2.841890862532807E+09,  9.274614570364714E+08,
-3.335603862250805E+07]) *1000
210 # Name des Himmelskörpers
211 data['Uranus']['name'] = 'Uranus'
212 # Farbe des Himmelskörpers
213 data['Uranus']['farbe'] = 'deepskyblue'
214 # Relative Grösse des Himmelskörpers
215 data['Uranus']['grösse'] = 9
216 # Endposition gemäss Ephemeride vom 23.09.2025 um 00:00 in AE
217 data['Uranus']['end_r'] = np.array([1.529034569238336E+09 *1000/au, 2.485654879037999E+09 *1000/
au,  -1.059476126538372E+07 *1000/au])
218
219
220 # Eintrag für den Neptun
221 data['Neptun'] = {}
222 # Masse
223 data['Neptun']['m'] = 1.02409e25
224 # Geschwindigkeit am 23.09.2015 um 00:00 (in x,y,z Richtung (kartesische Koordinaten) als
dreidimensionaler Array) in m/s
225 data['Neptun']['v'] = np.array([1.968709764849799E+00,  5.072291369137974E+00,
-1.493328241014094E-01]) *1000
226 # Position am 23.09.2015 um 00:00 (in x,y,z Richtung (kartesische Koordinaten) als
dreidimensionaler Array) in m
227 data['Neptun']['r'] = np.array([4.164599387125507E+09, -1.656186891604651E+09,
-6.186269297035909E+07]) *1000
```

```
228 # Name des Himmelskörpers
229 data['Neptun']['name'] = 'Neptun'
230 # Farbe des Himmelskörpers
231 data['Neptun']['farbe'] = 'mediumblue'
232 # Relative Grösse des Himmelskörpers
233 data['Neptun']['grösse'] = 8.5
234 # Endposition gemäss Ephemeride vom 23.09.2025 um 00:00 in AE
235 data['Neptun']['end_r'] = np.array([4.469783019533730E+09  *1000/au, 3.039168887282323E+07
*1000/au, -1.036283241921116E+08 *1000/au])
236
237
```

11 Literaturverzeichnis

[1] B. Högel, «biancahoegel.de,» 13. Juli 2024. [Online]. Available: https://www.biancahoegel.de/astronomie/zweikoerperproblem.html. [Zugriff am 9. Juli 2025].

[2] K. Stumpff, «Himmelsmechanik Band II,» in *Das Dreikörperproblem*, Berlin, VEB Deutscher Veralg der Wissenschaften, 1965, pp. 17-18.

[3] M. Bischoff, «Spektrum.de,» Spektrum der Wissenschaft Verlagsgesellschaft mbH, 3. November 2023. [Online]. Available: https://www.spektrum.de/kolumne/dreikoerperproblem-von-liu-cixin-sciencefiction-roman-zur-chaostheorie/2196210. [Zugriff am 27. August 2025].

[4] O. Jäger, «mein-lernen.at,» [Online]. Available: https://mein-lernen.at/pythagoras/pythagoras-gleichseitiges-dreieck/pythagoras-gleichseitiges-dreieck-2/. [Zugriff am 2. Januar 2026].

[5] L. Frischauf, «mathespass.at,» 2018. [Online]. Available: https://www.mathespass.at/formeln/gleichseitiges-dreieck-formeln-und-eigenschaften. [Zugriff am 2. Januar 2026].

[6] U. Gerlach, «The Ohio State University,» 9. Januar 2019. [Online]. Available: https://people.math.osu.edu/gerlach.1/math5756/5757lecturenotes/3-bodyProblem.pdf. [Zugriff am 27. Dezember 2025].

[7] F. Bruder, «Universität Hamburg,» 12. Februar 2007. [Online]. Available: https://www.math.uni-hamburg.de/home/gunesch/Vorlesung/WiSe2006-7/Sem_ODE/Materialien/Bruder/skript-Bruder.pdf. [Zugriff am 10. Oktober 2025].

[8] K. R. Meyer und G. R. Hall, Introduction to Hamiltonian Dynamical Systems and the N-Body Problem, New York: Springer-Verlag, 1992.

[9] M. Laine und T. Fliessbach, «Universität Bern,» 14. März 2022. [Online]. Available: http://laine.itp.unibe.ch/mechanikI/lec07.pdf. [Zugriff am 10. Oktober 2025].

[10] A. Müller, «Spektrum.de,» 2007. [Online]. Available: https://www.spektrum.de/lexikon/astronomie/newtonsche-gravitation/312. [Zugriff am 19. August 2025].

[11] E. Leitner und U. Finckh, «LEIFIphysik,» FWU Institut für Film und Bild in Wissenschaft und Unterricht gemeinnützige GmbH, [Online]. Available: https://www.leifiphysik.de/mechanik/gravitationsgesetz-und-feld/grundwissen/gravitationsgesetz-von-newton. [Zugriff am 19. August 2025].

[12] E. Leitner und U. Finckh, «LEIFIphysik,» FWU Institut für Film und Bild in Wissenschaft und Unterricht gemeinnützige GmbH, [Online]. Available: https://www.leifiphysik.de/mechanik/gravitationsgesetz-und-feld/geschichte/gravitationskonstante-historisch. [Zugriff am 19. August 2025].

[13] «Wikibooks,» Wikimedia Foundation Inc., 2. November 2023. [Online]. Available: https://de.wikibooks.org/wiki/Das_Mehrk%C3%B6rperproblem_in_der_Astronomie/_Allgemeine_L%C3%B6sungsmethoden/_Zwischenschritt-Verfahren:_Leapfrog_und_Runge-Kutta. [Zugriff am 9. Juli 2025].

[14] B. Tyrrell, «Trinitiy College Dublin,» 2017. [Online]. Available: https://www.maths.tcd.ie/~btyrrel/nbody.pdf. [Zugriff am 31. Januar 2025].

[15] W3Schools, «W3Schools,» Refsnes Data AS, W3Schools Network AS, [Online]. Available: https://www.w3schools.com/python/python_dictionaries.asp. [Zugriff am 7. August 2025].

[16] J. Hunter, «Matplotlib,» Juni 2007. [Online]. Available: https://matplotlib.org/stable/api/_as_gen/matplotlib.pyplot.figure.html. [Zugriff am 24. Dezember 2025].

[17] J. Hunter, «Matplotlib,» Juni 2007. [Online]. Available: https://matplotlib.org/stable/api/_as_gen/matplotlib.figure.Figure.add_subplot.html. [Zugriff am 24. Dezember 2025].

[18] J. Hunter, «Matplotlib,» Juni 2007. [Online]. Available: https://matplotlib.org/stable/api/_as_gen/matplotlib.axes.Axes.plot.html. [Zugriff am 24. Dezember 2025].

[19] J. Hunter, «Matplotlib,» Juni 2007. [Online]. Available: https://matplotlib.org/stable/gallery/color/named_colors.html#sphx-glr-gallery-color-named-colors-py. [Zugriff am 24. Dezember 2025].

[20] J. Hunter, «Matplotlib,» Juni 2007. [Online]. Available: https://matplotlib.org/stable/api/_as_gen/matplotlib.axes.Axes.legend.html. [Zugriff am 24. Dezember 2025].

[21] J. Hunter, «Matplotlib,» Juni 2007. [Online]. Available: https://matplotlib.org/stable/api/_as_gen/matplotlib.axes.Axes.set_title.html. [Zugriff am 24. Dezember 2025].

[22] J. Hunter, «Matplotlib,» Juni 2007. [Online]. Available: https://matplotlib.org/stable/api/_as_gen/matplotlib.axes.Axes.set_xlabel.html. [Zugriff am 24. Dezember 2025].

[23] J. Hunter, «Matplotlib,» Juni 2007. [Online]. Available: https://matplotlib.org/stable/api/_as_gen/matplotlib.pyplot.show.html. [Zugriff am 24.

Dezember 2025].

[24] J. Hunter, «Matplotlib,» Juni 2007. [Online]. Available: https://matplotlib.org/stable/gallery/mplot3d/wire3d_animation_sgskip.html. [Zugriff am 24. Dezember 2025].

[25] J. Hunter, «Matplotlib,» Juni 2007. [Online]. Available: https://matplotlib.org/stable/api/_as_gen/matplotlib.pyplot.cla.html. [Zugriff am 24. Dezember 2025].

[26] California Institute of Technology, «NASA Jet Propulsion Labaratory,» California Institute of Technology, 2. Juni 2025. [Online]. Available: https://ssd.jpl.nasa.gov/horizons/app.html#/. [Zugriff am 20. Oktober 2025].

[27] H. Bruderer, «ETH Zürich,» Mai 2011. [Online]. Available: https://ethz.ch/content/dam/ethz/special-interest/dual/educeth-dam/documents/Unterrichtsmaterialien/informatik/Zeitrechnung%20(Hintergrund)/Zeitrechnung_26.4.2011.pdf. [Zugriff am 8. November 2025].

[28] J. Hodge, «GitHub,» GitHub, Inc., 2024. [Online]. Available: https://github.com/jman4162/Three-Body-Problem-Simulator/blob/main/Three_Body_Problem_Simulator.ipynb?short_path=1b4a79c. [Zugriff am 16. November 2025].

[29] T. Ludyga, «Technische Universität Dresden,» 18, Oktober 2021. [Online]. Available: https://tud.qucosa.de/api/qucosa%3A77414/attachment/ATT-0/. [Zugriff am 9. September 2025].

[30] D. Lingenhöhl, «Spektrum der Wissenschaft,» Spektrum der Wissenschaft Verlagsgesellschaft mbH, 1998. [Online]. Available: https://www.spektrum.de/lexikon/physik/dreikoerperproblem/3389. [Zugriff am 27. Dezember 2025].

[31] California Institute of Technology, «NASA Earth Data,» California Institute of Technology, 19. Dezember 2025. [Online]. Available: https://www.earthdata.nasa.gov/data/space-geodesy-techniques/slr/orbit-predictions. [Zugriff am 27. Dezember 2025].

[32] N. J. Cornish, «NASA,» 1998. [Online]. Available: https://map.gsfc.nasa.gov/ContentMedia/lagrange.pdf. [Zugriff am 12. September 2025].

[33] N. Fischer, «Spektrum.de,» 2008. [Online]. Available: https://www.spektrum.de/sixcms/media.php/1308/Zentrales%20WiS%20Dokument%20SuW%201%202008.pdf. [Zugriff am 12. September 2025].

[34] F. Freistetter, «Spektrum.de,» 22. Dezember 2024. [Online]. Available:

https://www.spektrum.de/kolumne/das-jacobi-integral-und-die-lagrange-punkte/2246348. [Zugriff am 12. September 2025].

[35] M. Kemper und G. Willems, «Universität Münster,» 11. Januar 2012. [Online]. Available: https://www.uni-muenster.de/imperia/md/content/physik_ft/pdf/ws1112/seminar/111918/willems-kemper.pdf. [Zugriff am 28. August 2025].

[36] A. Barnett, «NASA Solar System Exploration,» National Aeronautics and Space Administration NASA, 5. Juni 2025. [Online]. Available: https://solarsystem.nasa.gov/planet-compare/. [Zugriff am 14. September 2025].

[37] N. J. Cornish, «NASA,» NASA, 4. September 2023. [Online]. Available: https://science.nasa.gov/resource/what-is-a-lagrange-point/. [Zugriff am 14. September 2025].

[38] D. Keil, «Abi-Physik,» [Online]. Available: https://www.abi-physik.de/buch/astronomie/kosmische-geschwindigkeiten/. [Zugriff am 10. Oktober 2025].

[39] J. Hunter, «matplotib,» Mai-Juni 2007. [Online]. Available: https://matplotlib.org/stable/gallery/color/named_colors.html#sphx-glr-gallery-color-named-colors-py. [Zugriff am 24. Dezember 2025].

12 Abbildungsverzeichnis

13 Tabellenverzeichnis