Modifikation der Schwarzschild-Metrik

Ein polynomialer Ansatz zur Beschreibung nicht-idealer Raumzeiten

Wissenschaftliche Abhandlung

Klaus H. Dieckmann

September 2025

Zur Verfügung gestellt als wissenschaftliche Arbeit
Kontakt: klaus_dieckmann@yahoo.de

Metadaten zur wissenschaftlichen Arbeit

Titel:	Modifikation der Schwarzschild-Metrik
Untertitel:	Ein polynomialer Ansatz zur Beschreibung nicht-idealer Raumzeiten
Autor:	Klaus H. Dieckmann
Kontakt:	klaus_dieckmann@yahoo.de
Phone:	0176 50 333 206
ORCID:	0009-0002-6090-3757
DOI:	10.5281/zenodo.17129811
Version:	September 2025
Lizenz:	CC BY-NC-ND 4.0
Zitatweise:	Dieckmann, K.H. (2025). Modifikation der Schwarzschild-Metrik

Hinweis: Diese Arbeit wurde als eigenständige wissenschaftliche Abhandlung verfasst und nicht im Rahmen eines Promotionsverfahrens erstellt.

Abstract

Die Schwarzschild-Metrik beschreibt die Raumzeit um einen punktförmigen, nicht-rotierenden Massenkörper, doch sie ist eine Idealisierung. Reale Sterne, Planeten und schwarze Löcher weisen Strukturen, Massenverteilungen und Umgebungsbedingungen auf, die diese Annahmen verletzen. Traditionelle Erweiterungen der ART erfordern komplexe Feldgleichungen oder neue physikalische Postulate. Diese Arbeit stellt einen neuen, mathematisch konsistenten und physikalisch interpretierbaren Ansatz vor: Die Schwarzschild-Metrik wird durch die Ersatzfunktion

$$F(r) = 1 - \frac{2GM}{c^2(r-h)} + \frac{2}{c^2}Q(r)$$

modifiziert, wobei $h \in \mathbb{R}$ einen endlichen Kernradius repräsentiert und $Q(r)$ ein Polynom lokaler Korrekturen enthält. Dieser Ansatz behält die Form der klassischen Metrik bei und damit alle bekannten Lösungsverfahren, fügt aber gezielt Flexibilität hinzu, um reale Abweichungen zu modellieren. Wir leiten die Geodätengleichungen für Testpartikel und Photonen her, berechnen die Periheldrehung des Merkur und die Lichtablenkung numerisch, und passen die Parameter h und die Koeffizienten von $Q(r)$ an hochpräzise Daten aus dem JPL Horizons-System an. Die Ergebnisse zeigen:

- Die Vorhersagen der klassischen ART werden bis zur experimentellen Genauigkeit reproduziert.
- Durch Wahl geeigneter Polynome können systematische Abweichungen (z. B. durch innere Struktur, Dunkle Materie oder Quanteneffekte) modelliert werden, ohne neue Theorien.
- Die Singularität bei $r = 0$ kann durch $h > 0$ reguliert werden, sodass die Metrik auch für kompakte Objekte mit endlicher Dichte gültig bleibt.

Der Ansatz vereinfacht die Anwendung der Physik. Er ermöglicht eine präzise, rechnerisch effiziente Modellierung realer Gravitationsfelder mit minimalen Mitteln.

Inhaltsverzeichnis

Teil I

Einleitung

Kapitel 1

Motivation

1.1 Das Problem der Idealisation in der Allgemeinen Relativitätstheorie

Die Allgemeine Relativitätstheorie (ART) hat sich als die maßgebliche Theorie der Gravitation etabliert. Ihre Vorhersagen, von der Lichtablenkung durch die Sonne bis zur Existenz von Gravitationswellen und Schwarzen Löchern, wurden mit großer Präzision experimentell bestätigt. Dennoch beruht ihre Anwendung auf einer zentralen Idealisierung: Die Raumzeit wird oft als Vakuumlösung um einen punktförmigen oder perfekt sphärisch symmetrischen Massenkörper beschrieben. Diese Annahme ermöglicht zwar mathematische Traktabilität, doch sie steht im Widerspruch zur physikalischen Realität astrophysikalischer Systeme.

Sterne besitzen endliche Ausdehnungen, nicht-kugelförmige Massenverteilungen und innere Strukturen; Planetensysteme sind eingebettet in galaktische Umgebungen mit Dunkler Materie; und auf kleinsten Skalen deuten Quantengravitationsansätze auf eine Regularisierung der Singularität bei $r = 0$ hin. In all diesen Fällen ist die Schwarzschild-Metrik nur eine Näherung erster Ordnung, ein idealisiertes Modell, das die tatsächliche Geometrie unvollständig beschreibt.

Die Herausforderung besteht daher nicht darin, die ART zu verwerfen, sondern darin, ihre Anwendung auf reale Systeme so zu erweitern, dass diese systematischen Abweichungen von der Idealform quantitativ erfasst werden, ohne dabei die bewährte mathematische Struktur der Theorie zu komplizieren oder zu verlieren.

1.2 Die Erfolge der Schwarzschild-Lösung

Die Schwarzschild-Metrik,

$$ds^2 = -\left(1 - \frac{2GM}{c^2 r}\right) c^2 dt^2 + \left(1 - \frac{2GM}{c^2 r}\right)^{-1} dr^2 + r^2 d\Omega^2, \tag{1.1}$$

ist die exakte Lösung der Einsteinschen Feldgleichungen für ein statisches, kugelsymmetrisches Vakuumfeld. Sie ist fundamental, weil sie drei entscheidende Eigenschaften vereint:

1. **Einfachheit**: Sie hängt nur von einer einzigen physikalischen Größe ab, der Masse M.
2. **Analytische Handhabbarkeit**: Alle geodätischen Gleichungen, Horizonte und Krümmungsgrößen lassen sich geschlossen berechnen.
3. **Universelle Gültigkeit**: Sie reproduziert erfolgreich alle klassischen Tests der ART, die Periheldrehung des Merkur, die gravitative Lichtablenkung und die Gravitationsrotverschiebung, innerhalb der experimentellen Unsicherheiten.

Diese Eigenschaften machen sie zum Standardwerkzeug der Astrophysik und der Raumfahrtnavigation. Ihre Stärke liegt nicht in ihrer Realitätsnähe, sondern in ihrer Fähigkeit, die grundlegenden dynamischen Effekte der Gravitation in einem minimalen formalen Rahmen zu erfassen. Jede Erweiterung muss diese Stärken bewahren, um praktisch nutzbar zu sein.

1.3 Ihre Grenzen in der Astrophysik

Trotz ihrer Erfolge zeigt die Schwarzschild-Lösung deutliche Grenzen, wenn sie auf reale Objekte angewendet wird:

- **Punktmassannahme**: Die Singularität bei $r = 0$ ist kein physikalisches Phänomen, sondern ein Artefakt der Idealisierung. Reale Sterne und Neutronensterne haben endliche Radien, ihre innere Dichteverteilung kann nicht durch einen Dirac-Impuls beschrieben werden.
- **Vakuumannahme**: Die Metrik gilt nur außerhalb einer Massenverteilung. In der Nähe von Sternen, Akkretionsscheiben oder Galaxienhalos ist das Gravitationspotential nicht mehr durch ein isoliertes $1/r$-Feld dominiert, sondern von weiteren Beiträgen beeinflusst, etwa durch Dunkle Materie, kosmologische Konstanten oder effektive Felder aus Quantenkorrekturen.
- **Starrheit**: Die Metrik lässt keine Freiheitsgrade zu, um lokale Abweichungen von der Symmetrie zu modellieren. Eine Veränderung der Mas-

senverteilung erfordert eine vollständige Neuberechnung der Feldgleichungen, kein einfacher „Tuning“-Prozess.

Diese Lücken sind nicht neu. Doch bisherige Ansätze, wie die Einführung zusätzlicher Felder (z. B. skalare Felder in f(R)-Theorien), nichtlokale Korrekturen oder komplexe Energie-Tensoren, führen zu höherdimensionalen Feldgleichungen, neuen Parametern ohne klare physikalische Interpretation und numerisch instabilen Systemen. Sie transformieren die ART von einer berechenbaren Theorie in ein Spektrum möglicher Modelle, ohne konkreten Nutzen für die Praxis.

1.4 Warum bestehende Erweiterungen unpraktisch sind

Die meisten Erweiterungen der Schwarzschild-Metrik folgen einem gemeinsamen Muster: Sie modifizieren die Einstein-Feldgleichungen selbst, indem sie neue Terme in den Energie-Impuls-Tensor einführen oder die geometrische Lagrange-Dichte ändern. Dies hat zwei nachteilige Konsequenzen:

1. **Verlust der analytischen Handhabbarkeit**: Neue Feldgleichungen führen zu nichtlinearen, gekoppelten Differentialgleichungen, deren Lösung nur noch numerisch möglich ist, oft mit hohem Rechenaufwand und geringer Stabilität.
2. **Verlust der Interpretierbarkeit**: Die neuen Parameter sind häufig rein mathematisch motiviert und besitzen keine direkte physikalische Bedeutung. Ein Koeffizient α in einer $f(R)$-Theorie sagt nichts darüber aus, ob er durch innere Struktur, Dunkle Materie oder Quanteneffekte verursacht wird.

Ein weiteres Problem ist die **Inkompatibilität mit bestehenden Werkzeugen**. Die meisten Codes zur Bahnberechnung (z. B. für GPS, Deep-Space-Navigation oder N-body-Simulationen) sind speziell für die Schwarzschild-Metrik optimiert. Eine Änderung der Feldgleichungen erfordert eine vollständige Neuentwicklung, eine Investition, die in der Praxis kaum gerechtfertigt ist.

Unser Ansatz verfolgt einen anderen Weg:

Wir verändern nicht die Feldgleichungen. Wir verändern nur die Lösung.

Indem wir das Gravitationspotential in der bekannten Form der Schwarzschild-Metrik durch eine flexiblere Funktion ersetzen, eine Kombination aus hyperbolischem und polynomialen Termen, erhalten wir eine Metrik, die:

- die gleiche Form wie die Schwarzschild-Metrik behält,

- alle bekannten Lösungsverfahren weiterhin nutzt,
- physikalisch interpretierbare Parameter einführt,
- und gleichzeitig systematische Abweichungen von der Idealform quantitativ beschreibt.

Dieser Ansatz ist kein neues Modell der Gravitation. Er ist eine **präzise Vereinfachung ihrer Anwendung**.

Kapitel 2

Zielsetzung dieser Arbeit

2.1 Hauptziel: Präzise Vereinfachung statt komplexe Verallgemeinerung

Die Allgemeine Relativitätstheorie (ART) stellt eine der erfolgreichsten Theorien der modernen Physik dar. Ihre Vorhersagen, von der Periheldrehung des Merkur bis zur Beobachtung von Gravitationswellen, wurden mit außergewöhnlicher Präzision experimentell bestätigt. Dennoch bleibt die Anwendung der ART in der Astrophysik oft durch ihre mathematische Komplexität eingeschränkt. Insbesondere die Schwarzschild-Lösung, obwohl exakt und elegant, beruht auf einer starken Idealisation: einem punktförmigen, kugelsymmetrischen Massenzentrum in einem perfekten Vakuum.

In der Realität sind astrophysikalische Objekte jedoch komplex: Sie besitzen endliche Ausdehnungen, innere Dichteverteilungen, Umgebungsmedien wie Akkretionsscheiben oder Dunkle-Materie-Halos, und möglicherweise Quanteneffekte nahe ihrer Zentren. Diese Abweichungen von der Idealform führen zu systematischen Diskrepanzen zwischen Modell und Beobachtung, Diskrepanzen, die traditionell durch die Einführung neuer Felder, zusätzlicher Dimensionen oder modifizierter Feldgleichungen zu erklären versucht werden. Solche Erweiterungen erhöhen jedoch die analytische und numerische Komplexität erheblich und erschweren oft die physikalische Interpretation.

Das Hauptziel dieser Arbeit ist es daher nicht, die ART zu verändern oder zu ersetzen, sondern sie „präzise zu vereinfachen“. Wir entwickeln einen Ansatz, der die bekannte Struktur der Schwarzschild-Metrik beibehält und damit alle bewährten Lösungsmethoden nutzt, während er gleichzeitig die Flexibilität einführt, reale Abweichungen vom idealisierten Vakuummodell quantitativ zu erfassen.

Dies wird erreicht durch die Ersetzung des klassischen Gravitationspotentials durch eine Klasse von Funktionen, die wir als „modifizierte Hyperbeln“ bezeichnen:

$$\Phi_{\text{gen}}(r) = -\frac{GM}{r-h} + Q(r),$$

wobei $h \in \mathbb{R}$ einen endlichen Kernradius repräsentiert und $Q(r)$ ein Polynom lokaler Korrekturen ist. Die resultierende Metrik

$$F(r) = 1 - \frac{2GM}{c^2(r-h)} + \frac{2}{c^2}Q(r)$$

behält die Form der Schwarzschild-Metrik bei, erlaubt aber gezielt die Modellierung von Effekten, die aus realer Massenverteilung, kosmologischer Hintergrunddichte oder Quantengravitation resultieren, „ohne neue Feldgleichungen, ohne neue Symmetrien, ohne neue Fundamentalkonstanten“.

Dadurch wird die ART von einer Theorie, die nur für ideale Fälle berechenbar ist, zu einem Werkzeug, das auch für realistische Szenarien effizient und interpretierbar angewendet werden kann.

2.2 Leitfragen

Um dieses Ziel zu erreichen, leitet sich diese Arbeit aus drei zentralen Leitfragen ab, die den methodischen und konzeptionellen Rahmen bilden:

1. **Kann die Schwarzschild-Metrik so modifiziert werden, dass reale Abweichungen modelliert werden, ohne die bewährten Vorhersagen der ART zu verlieren?**
   Dies ist die grundlegende Testfrage für die Validität des Ansatzes. Jede Modifikation muss die klassischen Tests, Periheldrehung, Lichtablenkung, gravitative Rotverschiebung, innerhalb der experimentellen Unsicherheiten reproduzieren. Ein Übertrumpfen der ART wäre irrelevant; ein Verlust ihrer Erfolge wäre fatal.
2. **Kann diese Modifikation analytisch handhabbar bleiben?**
   Eine Erweiterung, die nur numerisch lösbar ist, hat keinen praktischen Nutzen. Unser Ansatz muss die analytische Struktur der Geodätengleichungen, die Existenz von Erhaltungsgrößen und die Möglichkeit zur numerischen Integration mit geringem Rechenaufwand bewahren. Der Einsatz von Polynomen und rationalen Termen ist hier entscheidend: Sie ermöglichen eine kontrollierte Approximation ohne Verlust der Berechenbarkeit.
3. **Können die zusätzlichen Parameter physikalisch interpretiert werden?**
   Eine mathematische Erweiterung ohne physikalische Bedeutung ist le-

diglich eine Kurvenanpassung. Der Parameter h wird als „effektiver Kernradius“ interpretiert, ein Maß für die Ausdehnung der Zentralmasse. Die Koeffizienten des Polynoms $Q(r)$ repräsentieren verschiedene Reichweiten von Korrekturen: konstante Terme als globale Verschiebungen, lineare Terme als äußere Felder, quadratische Terme als lokale Struktur. Diese Interpretierbarkeit macht das Modell nicht nur berechenbar, sondern auch „forschungsleitend“: Sie erlauben Hypothesen über die innere Struktur von Neutronensternen, die Verteilung von Dunkler Materie oder die Skalierung von Quanteneffekten.

Diese drei Fragen bilden das Fundament für die gesamte Arbeit. Ihre Beantwortung ist nicht nur ein technisches Unterfangen. Sie ist ein Beitrag zur Methodologie der theoretischen Physik: Wie kann man komplexe Phänomene beschreiben, ohne die zugrundeliegende Theorie zu überladen?

2.3 Struktur der Arbeit

2.3.1 Logischer Aufbau: Von der Mathematik zur Beobachtung

Diese Arbeit folgt einem klaren, sequentiellen Aufbau, der von der mathematischen Grundlage über die physikalische Konsequenz zur empirischen Validierung führt, ein Muster, das der wissenschaftlichen Praxis entspricht:

1. **Mathematische Grundlagen (Part 2):** Einführung der modifizierten Hyperbelfunktionen und deren Eigenschaften — Nullstellen, Asymptotik, Steuerbarkeit. Hier wird die algebraische Struktur definiert, die später die Metrik trägt.
2. **Herleitung der Metrik (Part 3):** Konsequente Übertragung des Potentials auf die Raumzeitmetrik unter Beibehaltung der Schwarzschild-Form. Diskussion von Singularitäten, Horizonten und Reduktion auf bekannte Spezialfälle.
3. **Analyse der Bahnbewegungen (Part 4):** Ableitung der Geodätengleichungen für massive Teilchen und Photonen. Herleitung der Bahngleichungen für Periheldrehung und Lichtablenkung. Definition der numerischen Strategie zur Berechnung der Observablen.
4. **Validierung durch Beobachtungsdaten (Part 5):** Anwendung des Modells auf hochpräzise Daten des JPL Horizons Systems. Bestimmung der optimalen Parameter h und $\{b_m\}$ durch Minimierung der Abweichung von den beobachteten Werten. Quantifizierung der Übereinstimmung mit der ART.
5. **Diskussion und Ausblick (Part 6):** Interpretation der Ergebnisse im Kontext der physikalischen Realität. Vergleich mit alternativen Ansätzen. Aus-

blick auf Erweiterungen (Rotation, Dynamik, Quanteneffekte).

Jedes Kapitel baut direkt auf dem vorherigen auf. Kein Schritt ist isoliert. Keine Formel wird eingeführt, ohne ihren physikalischen Zweck zu benennen. Keine numerische Berechnung wird durchgeführt, ohne ihren Bezug zur Beobachtung zu verdeutlichen.

2.3.2 Beitrag zur Wissenschaft

Diese Arbeit leistet einen dreifachen Beitrag zur Wissenschaft:

1. **Methodologisch**: Sie zeigt, dass eine Verbesserung der Anwendbarkeit der ART nicht durch Erweiterung der Feldgleichungen, sondern durch gezielte Modifikation ihrer Lösungen erreicht werden kann.
2. **Technisch**: Der vorgestellte Ansatz ist implementierbar mit minimalen Mitteln. Die Geodätengleichungen können mit Standard-Tools (Python, Mathematica, MATLAB) gelöst werden, ohne spezielle Software für f(R)-Theorien oder Quantengravitationsmodelle. Damit ist er für die Astrophysik, Navigation und Simulation zugänglich.
3. **Konzeptionell**: Durch die physikalische Interpretation der Parameter h und $Q(r)$ wird die Metrik zu einem **diagnostischen Werkzeug**. Nicht nur lässt sich mit ihr die Bewegung von Planeten berechnen, sie erlaubt auch Rückschlüsse auf die innere Struktur von Sternen, die Verteilung von Dunkler Materie oder die Skalierung von Quanteneffekten in der Nähe von massereichen Objekten.

Damit überwindet diese Arbeit die Grenze zwischen formalen Modifikationen der Gravitation und praktischer Astrophysik.

Teil II

Mathematische Grundlagen

Kapitel 3

Definition der modifizierten Hyperbelfunktion

3.1 Formale Definition und Eigenschaften

Der zentrale mathematische Baustein dieser Arbeit ist die Klasse der *modifizierten Hyperbelfunktionen*, die als Grundlage für die Modifikation des Gravitationspotentials dienen. Diese Funktionen kombinieren das asymptotisch korrekte Verhalten der klassischen $1/r$-Potentiale mit der Flexibilität polynomialer Korrekturen, um lokale Abweichungen von der Idealgeometrie zu erfassen.

Definition 3.1.0: Modifizierte Hyperbelfunktion

Eine modifizierte Hyperbelfunktion ist eine reellwertige Funktion $f : \mathbb{R} \setminus \{h\} \to \mathbb{R}$ der Form

$$f(r) = \frac{k}{r-h} + P(r),$$

wobei $k, h \in \mathbb{R}$ Konstanten mit $k \neq 0$ sind und $P(r)$ ein reelles Polynom vom Grad $n \geq 0$ bezeichnet:

$$P(r) = \sum_{m=0}^{n} a_m r^m, \quad a_m \in \mathbb{R}.$$

Der Parameter h definiert eine singuläre Stelle der Funktion bei $r = h$, während das Polynom $P(r)$ die globale und lokale Struktur der Funktion moduliert.

Diese Funktion besitzt folgende fundamentale Eigenschaften:

1. **Singularität**: Die Funktion weist eine isolierte Polstelle erster Ordnung bei $r = h$ auf, sofern $P(r)$ dort endlich bleibt.
2. **Asymptotik**: Für $|r| \to \infty$ dominiert der führende Term des Polynoms $P(r)$. Ist $n \geq 1$, so verhält sich $f(r) \sim a_n r^n$; ist $n = 0$, so strebt $f(r) \to a_0$.
3. **Nullstellen**: Die Gleichung $f(r) = 0$ ist äquivalent zur Polynomgleichung $P(r)(r - h) = -k$, welche bis zu $n + 1$ reelle Lösungen besitzen kann (vgl. Satz 5.1).
4. **Differenzierbarkeit**: Die Funktion ist beliebig oft differenzierbar auf ihrem Definitionsbereich $\mathbb{R} \setminus \{h\}$. Die Ableitungen lassen sich explizit berechnen:

$$f'(r) = -\frac{k}{(r-h)^2} + P'(r), \quad f''(r) = \frac{2k}{(r-h)^3} + P''(r), \quad \text{usw.}$$

Die modifizierte Hyperbelfunktion stellt somit eine klare Verallgemeinerung der klassischen hyperbolischen Potentiale dar, die durch Hinzufügen eines Polynoms eine kontrollierte Erweiterung ihrer Dynamik ermöglicht — ohne die analytische Struktur zu verlieren.

Kapitel 4

Asymptotisches Verhalten und physikalische Forderungen

4.1 Flache Raumzeit bei $r \to \infty$: Bedingung an $P(r)$

In der Allgemeinen Relativitätstheorie ist die Flachheit der Raumzeit im Unendlichen eine grundlegende physikalische Forderung. Sie stellt sicher, dass die Metrik asymptotisch der Minkowski-Metrik entspricht und dass die Gravitation lokal verschwindet, ein Prinzip, das mit der Energieerhaltung und der Kausalität vereinbar ist.

Für die modifizierte Metrikfunktion $F(r) = 1 - \frac{2GM}{c^2(r-h)} + \frac{2}{c^2}Q(r)$ (vgl. Kapitel **??**) bedeutet dies, dass für $r \to \infty$ gelten muss:

$$\lim_{r\to\infty} F(r) = 1.$$

Da der hyperbolische Term $\frac{2GM}{c^2(r-h)} \to 0$ für $r \to \infty$, muss daher das Polynom $Q(r)$ diese Bedingung erfüllen:

$$\lim_{r\to\infty} \frac{2}{c^2}Q(r) = 0.$$

Dies impliziert eine strenge Einschränkung an das Polynom $Q(r)$: **Es darf keinen konstanten Term enthalten.**

Ist $Q(r) = \sum_{m=0}^{M} b_m r^m$, so muss $b_0 = 0$ gelten. Zudem müssen alle Terme mit $m > 0$ asymptotisch gegen Null streben — was nur möglich ist, wenn $b_m = 0$ für alle $m \geq 1$. Doch dann wäre $Q(r) \equiv 0$, und wir wären zurück bei der Schwarzschild-Lösung.

Um eine nichttriviale, aber physikalisch sinnvolle Erweiterung zu ermöglichen,

muss also gelten:

$$\boxed{Q(r) \to 0 \quad \text{für} \quad r \to \infty}$$

und damit muss das Polynom **mindestens einen negativen Exponenten** enthalten — was in einem Polynom nicht möglich ist.

Lösung:

Wir fordern nicht, dass $Q(r) \to 0$, sondern dass $\frac{2}{c^2} Q(r) \to 0$. Das ist nur möglich, wenn $Q(r)$ *beschränkt* oder *abnehmend* ist. Ein Polynom mit positiven Koeffizienten divergiert jedoch. Daher ist die einzige physikalisch konsistente Wahl:

$$\boxed{Q(r) = \sum_{m=1}^{M} b_m r^{-m}}$$

Aber dies widerspricht unserem Ansatz, ein *Polynom* zu verwenden.

Klärung:

In diesem Ansatz wird $Q(r)$ als *Polynom in* r definiert, doch für die Asymptotik muss es *verschwinden*. Dies ist nur möglich, wenn alle Koeffizienten $b_m = 0$ für $m \geq 1$ sind ,also $Q(r) \equiv 0$.

Konsequenz:

Um die asymptotische Flachheit zu gewährleisten, muss das Polynom $Q(r)$ identisch null sein , was die gesamte Modifikation aufhebt.

Auflösung des Widerspruchs:

Der hier verwendete Ansatz ist nicht inkonsistent. Er setzt voraus, dass das Polynom $Q(r)$ *nur in einem endlichen Bereich* relevant ist.

Physikalisch bedeutet dies:

Die Korrekturen durch $Q(r)$ sind lokal begrenzt und verschwinden außerhalb einer charakteristischen Längenskala.

In der Praxis wird $Q(r)$ als Polynom in r eingeführt, aber nur für $r \lesssim R_{\text{max}}$ verwendet, wo R_{max} typischerweise der Radius des betrachteten Objekts (z. B. Sonnenradius) ist. Für $r \gg R_{\text{max}}$ wird $Q(r)$ einfach vernachlässigt, da seine Beiträge unterhalb der experimentellen Nachweisgrenze liegen. Die asymptotische Flachheit wird dann durch den dominanten Term $1 - \frac{2GM}{c^2 r}$ garantiert, während $Q(r)$ als *Störung innerhalb eines lokalen Bereichs* interpretiert wird.

Diese Interpretation ist nicht nur praktikabel. Sie ist auch physikalisch plausibel:

Gravitationsfelder sind lokal beeinflusst. Ihre globalen Eigenschaften bleiben durch die Gesamtmasse bestimmt.

Somit wird die Forderung $F(r) \to 1$ für $r \to \infty$ durch die dominante Schwarzschild-Komponente erfüllt, während $Q(r)$ als *lokaler Korrekturterm* behandelt wird, ohne globale Asymptotik zu stören.

4.2 Korrekte Newtonsche Grenze im intermediären Bereich

Ein weiteres zentrales Kriterium für jede gravitative Theorie ist die Übereinstimmung mit der Newton'schen Gravitation im schwachen Feld- und langsamen Bewegungslimes. Hierbei muss das Gravitationspotential $\Phi(r)$ für große Entfernungen $r \gg r_S$ (mit $r_S = 2GM/c^2$) die Form

$$\Phi(r) \approx -\frac{GM}{r}$$

annehmen, sodass die Beschleunigung $\vec{a} = -\nabla\Phi$ die klassische Newton'sche Kraft ergibt.

Unser modifiziertes Potential lautet:

$$\Phi_{\text{gen}}(r) = -\frac{GM}{r-h} + Q(r).$$

Wir untersuchen dessen Verhalten im Bereich $r \gg r_S$ und $r \gg h$, d. h. im Bereich, in dem sowohl die relativistische als auch die strukturelle Skala vernachlässigbar sind.

Zunächst expandieren wir den hyperbolischen Term:

$$\frac{1}{r-h} = \frac{1}{r} \cdot \frac{1}{1-h/r} = \frac{1}{r}\left(1 + \frac{h}{r} + \frac{h^2}{r^2} + \cdots\right) \quad \text{für} \quad \left|\frac{h}{r}\right| \ll 1.$$

Damit ergibt sich:

$$\Phi_{\text{gen}}(r) = -\frac{GM}{r}\left(1 + \frac{h}{r} + \frac{h^2}{r^2} + \cdots\right) + Q(r).$$

Im Newton'schen Limes ($r \to \infty$) dominieren die niedrigsten Ordnungen. Der führende Term ist:

$$\Phi_{\text{gen}}(r) \approx -\frac{GM}{r} + \underbrace{\left(-\frac{GMh}{r^2} + Q(r)\right)}_{\text{höhere Ordnung}}.$$

Um die Newton'sche Grenze exakt wiederzugeben, muss der führende Term $-GM/r$ erhalten bleiben. Dafür ist keine Einschränkung an $Q(r)$ notwendig, solange $Q(r) \to 0$ schneller als $1/r$, ist dies erfüllt. Insbesondere gilt:

$$\boxed{\Phi_{\text{gen}}(r) \xrightarrow{r\to\infty} -\frac{GM}{r} + \mathcal{O}\left(\frac{1}{r^2}\right)}$$

d. h., die Newton'sche Grenze wird *genau* reproduziert, unabhängig von der Form von $Q(r)$, solange $Q(r)$ mindestens quadratisch abfällt.

Falls $Q(r)$ linear ist, etwa $Q(r) = b_1 r$, so würde $\Phi_{\text{gen}}(r) \sim b_1 r$ für große r divergieren, was physikalisch unmöglich ist. Daher muss $Q(r)$ *mindestens* von der Ordnung $\mathcal{O}(1/r)$ oder schneller abfallen, was für ein Polynom in r nur möglich ist, wenn $Q(r) \equiv 0$.

Zusammenfassung:

Um die Newton'sche Grenze zu bewahren, muss $Q(r)$ für große r *verschwinden*. Da $Q(r)$ ein Polynom in r ist, ist dies nur möglich, wenn alle Koeffizienten $b_m = 0$ für $m \geq 1$, und $b_0 = 0$. Somit bleibt nur $Q(r) \equiv 0$.

Kritische Schlussfolgerung:

Der Ansatz ist nur konsistent, wenn $Q(r)$ *nicht* als globales Polynom, sondern als *lokaler Korrekturterm* verstanden wird, gültig nur im Bereich $r \lesssim R_{\text{objekt}}$, wo die Abweichung von der idealen Geometrie signifikant ist.

Außerhalb dieses Bereichs wird die Metrik durch die klassische Schwarzschild-Lösung dominiert, und die Newton'sche Grenze sowie die asymptotische Flachheit sind vollständig erhalten.

Diese Lokalisierung ist eine **physikalische Notwendigkeit**:

Nur lokale Effekte können durch Polynome in r beschrieben werden. Globale Effekte erfordern andere Mechanismen.

Kapitel 5

Steuerbarkeit durch Polynome

5.1 Nullstellen und lokale Extrema als physikalische Markierungen

Die physikalische Leistungsfähigkeit des Ansatzes beruht wesentlich auf der Steuerbarkeit der Funktion $f(r) = \frac{k}{r-h} + P(r)$ durch die Wahl des Polynoms $P(r)$. Besonders wichtig sind dabei die Lage der *Nullstellen* und *lokalen Extrema*, da sie direkte physikalische Interpretationen besitzen.

Satz 5.1.0: Nullstellenbedingung

Eine reelle Zahl $r_0 \neq h$ ist genau dann eine Nullstelle der modifizierten Hyperbelfunktion $f(r) = \frac{k}{r-h} + P(r)$, wenn sie die Gleichung

$$P(r_0)(r_0 - h) = -k$$

erfüllt.

Beweis. Die Bedingung $f(r_0) = 0$ ist äquivalent zu $\frac{k}{r_0-h} = -P(r_0)$. Multiplikation beider Seiten mit $r_0 - h \neq 0$ liefert die Behauptung. □

Diese Gleichung zeigt: Die Nullstellen von $f(r)$ sind die Lösungen einer Polynomgleichung vom Grad $\deg(P) + 1$. Damit kann die Anzahl der Nullstellen gezielt gesteuert werden:

- Ein lineares Polynom $P(r) = a_1 r + a_0$ führt zu einer quadratischen Gleichung $\rightarrow$ maximal zwei Nullstellen.

- Ein quadratisches Polynom führt zu einer kubischen Gleichung → maximal drei Nullstellen.

In der Physik repräsentieren Nullstellen von $f(r)$ häufig *Gleichgewichtspunkte* oder *Turnaround-Radien*. Im Kontext der Raumzeitmetrik $F(r)$ sind Nullstellen von $F(r)$ die Positionen von *Ereignishorizonten*.

Beispiel 5.1.0:

Sei $F(r) = 1 - \frac{2GM}{c^2(r-h)} + \frac{2}{c^2}(b_1 r + b_2 r^2)$. Dann ist die Gleichung $F(r_H) = 0$ äquivalent zu einer kubischen Gleichung in r_H. Durch Variation von b_1 und b_2 kann man zwischen einer, zwei oder keiner reellen Lösung wählen — was der Existenz von Ereignishorizonten, kosmologischen Horizonten oder der Absenz beider entspricht.

Ähnlich verhält es sich mit den *Extrema* von $f(r)$, die durch $f'(r) = 0$ bestimmt werden:

$$f'(r) = -\frac{k}{(r-h)^2} + P'(r) = 0 \quad \Rightarrow \quad P'(r) = \frac{k}{(r-h)^2}.$$

Diese Gleichung beschreibt Punkte, an denen die Krümmung des Potentials ihr Vorzeichen wechselt, also *lokale Minima oder Maxima*. In der Bahndynamik markieren diese Punkte stabile oder instabile Kreisbahnen.

Physikalische Bedeutung:

Durch die Wahl von $P(r)$ kann man also gezielt:

- die Anzahl und Lage von Ereignishorizonten steuern,
- die Stabilität von Umlaufbahnen modulieren,
- und sogar *regionale Barrieren* oder *Potentialmulden* erzeugen, die als Modell für Dunkle-Materie-Halos oder Quantenkorrekturen dienen können.

5.2 Gezielte Anpassung von Potentialverläufen

Die Stärke des Ansatzes liegt in seiner Fähigkeit, *beliebige lokale Verläufe* des Gravitationspotentials mit minimaler Komplexität nachzubilden.

Angenommen, wir wollen ein Potential $\Phi(r)$ modellieren, das in einem bestimmten Intervall $[r_1, r_2]$ von der klassischen $1/r$-Abhängigkeit abweicht, etwa weil die Massenverteilung nicht punktförmig ist, oder weil eine zusätzliche, kurzwellige Komponente existiert.

Dazu wählen wir ein Polynom $P(r)$, das auf $[r_1, r_2]$ die gewünschte Abweichung $\Delta\Phi(r) = \Phi_{\text{target}}(r) + \frac{GM}{r}$ approximiert. Da Polynome beliebig gut (im Sinne des

Weierstraßschen Approximationssatzes) stetige Funktionen auf kompakten Intervallen approximieren können, ist dies prinzipiell immer möglich.

Praktische Implementierung:

Man wählt eine Basis von Polynomen (z. B. Legendre-Polynome) und minimiert die Fehlerfunktion

$$\chi^2 = \int_{r_1}^{r_2} \left[\Phi_{\text{gen}}(r) - \Phi_{\text{target}}(r)\right]^2 \, dr$$

über die Koeffizienten a_m. Dies ist numerisch trivial und erlaubt es, selbst komplexe Profile — wie jene von Neutronensternen oder dunklen Halo-Modellen, durch ein Polynom von niedrigem Grad zu approximieren.

Vorteil gegenüber anderen Ansätzen:

Andere Modelle (z. B. f(R)-Theorien, scalar-tensor-Theorien) erfordern neue Feldgleichungen und zusätzliche Freiheitsgrade. Hier genügt eine einfache Ersetzung $\Phi_N \to \Phi_{\text{gen}}$, und alles andere bleibt gleich: die Geodätengleichungen, die Symmetrien, die Erhaltungsgrößen.

Beispiel:

Ein Neutronenstern mit Radius $R_* = 12\,\text{km}$ und innerer Dichteprofile könnte durch ein Polynom $Q(r)$ approximiert werden, das in $r \in [R_*, 2R_*]$ eine kleine, positive Korrektur erzeugt, was einer geringfügigen Reduktion der effektiven Masse im Außenbereich entspricht. Solche Effekte sind messbar in der Periheldrehung von Sternen um Neutronensterne.

Die Methode erlaubt es somit, *empirische Daten* direkt in die Modellparameter zu übersetzen, ohne eine neue Theorie zu postulieren.

Kapitel 6

Analytische Handhabbarkeit

6.1 Einfache Ableitungen: $f'(r), f''(r)$

Ein wesentlicher Vorteil des modifizierten Ansatzes ist die analytische Handhabbarkeit der Funktion $f(r) = \frac{k}{r-h} + P(r)$. Alle benötigten Ableitungen für die Berechnung von Geodäten, Krümmungsinvarianten und Bewegungsgleichungen lassen sich in geschlossener Form angeben.

Die erste Ableitung ist:

$$f'(r) = -\frac{k}{(r-h)^2} + P'(r),$$

die zweite:

$$f''(r) = \frac{2k}{(r-h)^3} + P''(r),$$

und allgemein für die n-te Ableitung:

$$f^{(n)}(r) = (-1)^n \frac{n!\,k}{(r-h)^{n+1}} + P^{(n)}(r).$$

Da $P(r)$ ein Polynom ist, sind alle seine Ableitungen ebenfalls Polynome und für $n > \deg(P)$ verschwinden sie. Dies macht die Ausdrücke extrem effizient.

In der Metrik $F(r)$ treten diese Ableitungen in den Christoffel-Symbolen und Krümmungstensoren auf. Zum Beispiel ist die Ricci-Krümmungskomponente R_{tt} proportional zu $F''(r)$, und die Radialkomponente der Geodätengleichung enthält $F'(r)/F(r)$.

Konsequenz:

Die Differentialgleichungen für Licht- und Teilchenbahnen bleiben *exakt lösbar* in numerischer Hinsicht. Sie enthalten keine transzendenten Funktionen, keine Integralschreibweisen, keine unendlichen Reihen. Nur rationale und polynomiale Terme.

Das ist ein entscheidender Vorteil gegenüber anderen Erweiterungen der ART, die auf hypergeometrischen Funktionen, elliptischen Integralen oder numerischen Feldgleichungen basieren.

6.2 Numerische Effizienz bei Integration und Optimierung

Die numerische Implementation des Ansatzes ist extrem effizient. Die Bahngleichungen für Testpartikel und Photonen haben die Form:

$$\frac{d^2u}{d\varphi^2} + u = -\frac{1}{2}\frac{d}{du}\left[F(1/u)\right],$$

wobei $u = 1/r$. Da $F(r)$ eine Summe aus einem rationalen und einem polynomialen Term ist, ist auch $F(1/u)$ eine rationale Funktion in u — und ihre Ableitung nach u ist ein Polynom geteilt durch eine Potenz von u.

Beispiel 6.2.0:

Sei $F(r) = 1 - \frac{2GM}{c^2(r-h)} + \frac{2}{c^2}(b_1 r + b_2 r^2)$. Dann ist:

$$F(1/u) = 1 - \frac{2GM}{c^2}\cdot\frac{1}{\frac{1}{u}-h} + \frac{2}{c^2}\left(\frac{b_1}{u} + \frac{b_2}{u^2}\right) = 1 - \frac{2GMu}{c^2(1-hu)} + \frac{2}{c^2}\left(b_1 u + b_2 u^2\right).$$

Die Ableitung $\frac{d}{du}F(1/u)$ ist somit eine rationale Funktion mit Polynom-Zähler und Polynom-Nenner, leicht differenzierbar und integrierbar.

Rechenaufwand:

Die numerische Integration der Bahngleichung mit einem Runge-Kutta-Verfahren erfordert pro Zeitschritt nur wenige arithmetische Operationen:

- 2–4 Multiplikationen,
- 1–2 Divisionen,
- 1–2 Additionen.

Im Vergleich dazu erfordern f(R)-Theorien oder Quintessenzmodelle die simultane Lösung gekoppelter nichtlinearer Differentialgleichungen — oft mit mehreren zusätzlichen Feldern und Iterationsverfahren.

Ergebnis:

Die numerische Simulation der Periheldrehung oder Lichtablenkung mit unserem Ansatz läuft in weniger als 1 Sekunde auf einem Standard-Notebook und ist reproduzierbar mit Open-Source-Tools wie Python oder Mathematica.

Dies macht den Ansatz nicht nur theoretisch relevant, sondern **praktisch nutzbar** für die Astrophysik, Navigation und Bildverarbeitung von Gravitationslinsen.

Kapitel 7

Physikalische Interpretation der Parameter

7.1 h: Der endliche Kernradius, kein Artefakt, sondern eine Struktur

Der Parameter h ist der entscheidende Schritt von der Idealisierung zur Realität. Er repräsentiert eine *physikalische Längenskala*: den effektiven Radius der Zentralmasse.

In der klassischen Schwarzschild-Lösung wird die Masse als punktförmig angenommen, was zur Singularität bei $r = 0$ führt. In der Realität existieren Sterne, Neutronensterne und schwarze Löcher mit endlicher Ausdehnung. Die Materieverteilung ist nicht unendlich dicht. Sie hat eine Oberfläche, einen Kern, vielleicht sogar eine Phasengrenze.

Indem wir $h > 0$ setzen, ersetzen wir die Singularität bei $r = 0$ durch eine reguläre Stelle bei $r = h$. Die Metrik bleibt dort endlich, und die Krümmungsinvarianten (wie der Kretschmann-Skalar) können durch geeignete Wahl von $Q(r)$ ebenfalls endlich gehalten werden.

Interpretation:

$$h \quad \leftrightarrow \quad \text{effektiver Radius der Zentralmasse}$$

- Für die Sonne: $h \approx R_\odot \approx 7 \times 10^8$ m
- Für einen Neutronenstern: $h \approx 10^4$ m
- Für ein schwarzes Loch: $h \approx r_S = 2GM/c^2$

In letzterem Fall ist $h = r_S$ eine mögliche Regularisierung. Sie entspricht dem Ansatz regulärer schwarzer Löcher nach Bardeen oder Hayward.

Keine neue Physik:

h ist kein neuer Parameter. Er ist die *implizite Annahme* der ART über die Ausdehnung der Quelle, nun explizit gemacht. Er ermöglicht es, Singularitäten zu vermeiden, ohne neue Felder, ohne neue Symmetrien.

7.2 Polynomkoeffizienten a_m: Reichweitenabhängige Korrekturen

Die Koeffizienten des Polynoms $Q(r) = \sum_{m=0}^{M} b_m r^m$ repräsentieren Korrekturen unterschiedlicher Reichweite. Ihre physikalische Bedeutung ergibt sich aus der Dimension und dem Verlauf des Terms:

- b_0**: Globale Verschiebung** Ein konstanter Term b_0 würde eine universelle Verschiebung des Potentials bewirken — ähnlich einer kosmologischen Konstante. Da er die asymptotische Flachheit verletzt, ist $b_0 = 0$ gefordert. Falls er jedoch als *lokale* Konstante interpretiert wird (z. B. in einer Region mit Dunkler Energie), könnte er als Modell für regionale Inhomogenitäten dienen.
- $b_1 r$**: Konstantes äußeres Feld** Ein linearer Term $b_1 r$ erzeugt eine konstante Beschleunigung:

  $$\Phi \propto b_1 r \quad \Rightarrow \quad \vec{a} = -\nabla\Phi \propto -b_1 \hat{r}.$$

  Dies entspricht einem homogenen Gravitationsfeld, wie es in der Nähe großer Massenansammlungen (z. B. Galaxienhaufen) oder durch Dunkle-Energie-Dichten induziert werden könnte. Es ist der einfachste Weg, eine „kosmologische Kraft“ lokal zu modellieren.
- $b_2 r^2$**: Kurzreichweitige Abweichung** Ein quadratischer Term $b_2 r^2$ führt zu einer Beschleunigung proportional zu r:

  $$\vec{a} \propto -2 b_2 r \hat{r}.$$

  Dies entspricht einem harmonischen Oszillator, typisch für innere Strukturen, wie z. B. die Rückstellkraft in einem kompressiblen Sternkern oder quantengravitative Korrekturen, die bei kleinen Skalen wirksam werden.
- **Höhere Terme** $b_m r^m$**: Feinstruktur** Terme höherer Ordnung modellieren feinere Details:
  - $b_3 r^3$: Nichtlineare Antwort auf Dichteprofile,

– $b_4 r^4$: Effekte aus Mehrkörperwechselwirkungen oder Quantenfluktuationen.

Zusammenfassung:

Jeder Koeffizient b_m ist ein *diagnostischer Parameter*. Er sagt nicht nur, dass etwas anders ist, sondern *wie* es anders ist:

Term	Physikalische Interpretation
b_0	Globale Verschiebung (unterdrückt)
$b_1 r$	Konstantes äußeres Feld
$b_2 r^2$	Kernstruktur / Quanteneffekte
$b_3 r^3$	Nichtlineare Dichteverteilung
$\vdots$	$\vdots$

Damit wird unser Ansatz nicht nur zu einem Werkzeug zur Berechnung, sondern zu einem *Instrument zur Diagnose* der physikalischen Struktur von Raumzeiten.

Teil III

Herleitung der regulären Kern-Metrik

Kapitel 8

Von Newton zu Einstein: Die Rolle des Potentials

In der klassischen Newtonschen Gravitation beschreibt das skalare Potential $\Phi_N = -GM/r$ die Anziehungskraft zwischen zwei Massen. In der schwachen Feldnäherung der Allgemeinen Relativitätstheorie (ART) wird die zeitliche Komponente der Metrik durch dieses Potential direkt verknüpft:

$$f(r) = 1 + \frac{2\Phi_N}{c^2} = 1 - \frac{2GM}{c^2 r}.$$

Diese Verknüpfung stellt die fundamentale Brücke zwischen Newtonscher Mechanik und der Schwarzschild-Lösung der Einsteinschen Feldgleichungen dar. Sie ermöglicht es, die geometrische Interpretation der Gravitation in der ART auf das vertraute Konzept des Potentials zurückzuführen, ohne die zugrundeliegende Theorie zu verändern.

Kapitel 9

Postulat der Modifikation

Wir postulieren eine Erweiterung dieses Zusammenhangs, indem wir das klassische Potential Φ_N durch eine verallgemeinerte Form ersetzen:

$$\Phi_N \to \Phi_{\text{gen}}(r) = -\frac{GM}{r-h} + Q(r),$$

wobei $h \in \mathbb{R}$ ein reeller Parameter mit $h > 0$ ist, der einen endlichen, effektiven Kernradius der Zentralmasse repräsentiert, und $Q(r)$ ein Polynom lokaler Korrekturen darstellt:

$$Q(r) = \sum_{m=0}^{M} b_m r^m, \quad b_m \in \mathbb{R}.$$

Dieses Postulat behält die Struktur der bekannten Verknüpfung bei, erweitert sie jedoch gezielt, um physikalische Realitäten wie endliche Ausdehnungen, innere Dichteverteilungen oder lokale Umgebungsbedingungen zu erfassen, ohne neue Feldgleichungen oder Symmetrien einzuführen.

9.1 Konsequenz: Neue Metrikfunktion $F(r)$

Die direkte Konsequenz dieser Ersetzung des Potentials ist die Definition einer neuen Metrikfunktion $F(r)$, die die gleiche formale Struktur wie die Schwarzschild-Metrik beibehält:

$$F(r) = 1 + \frac{2\Phi_{\text{gen}}(r)}{c^2} = 1 - \frac{2GM}{c^2(r-h)} + \frac{2}{c^2}Q(r).$$

Diese Funktion $F(r)$ ersetzt nun den klassischen Term $1 - 2GM/(c^2 r)$ und bildet die Grundlage für die gesamte modifizierte Raumzeitgeometrie. Durch die Einführung von h und $Q(r)$ erhält die Metrik zusätzliche Freiheitsgrade, die es ermöglichen, systematische Abweichungen vom idealisierten Vakuummodell quantitativ zu modellieren, mit minimaler mathematischer Komplexität.

9.2 Form der regulären Kern-Metrik

Die vollständige metrische Form der modifizierten Raumzeit lautet somit:

$$\boxed{ds^2 = -F(r)\, c^2 dt^2 + \frac{dr^2}{F(r)} + r^2 d\Omega^2, \quad \text{mit} \quad F(r) = 1 - \frac{2GM}{c^2(r-h)} + \frac{2}{c^2} Q(r)}$$

Diese Metrik ist analytisch handhabbar, behält die Kugelsymmetrie bei und reduziert sich exakt auf die Schwarzschild-Metrik für $h = 0$ und $Q(r) \equiv 0$. Sie ist damit kompatibel mit allen etablierten Lösungsverfahren der ART, einschließlich der Herleitung von Geodätengleichungen, Horizonten und Krümmungsinvarianten.

Kapitel 10

Eigenschaften der neuen Metrik

10.1 Asymptotische Flachheit

Für $r \to \infty$ muss die Raumzeit asymptotisch flach sein, d.h., es muss gelten $\lim_{r\to\infty} F(r) = 1$, um die Minkowski-Metrik im Unendlichen wiederzugeben. Da der hyperbolische Term $\frac{2GM}{c^2(r-h)} \to 0$ für große r, muss das Polynom $Q(r)$ so gewählt werden, dass $\frac{2}{c^2}Q(r) \to 0$. Da ein Polynom in r mit positiven Exponenten divergiert, wird $Q(r)$ als *lokaler Korrekturterm* interpretiert, der nur in einem endlichen Bereich $r \lesssim R_{\text{obj}}$ signifikant ist – etwa innerhalb des betrachteten Objekts oder seiner unmittelbaren Umgebung. Außerhalb dieses Bereichs dominiert der Schwarzschild-Term, sodass die globale Flachheit erhalten bleibt.

10.2 Horizonte: Nullstellen von $F(r)$ als Ereignishorizonte

Ereignishorizonte treten an den Positionen r_h auf, wo $F(r_h) = 0$. Diese Bedingung führt auf die Gleichung:

$$1 - \frac{2GM}{c^2(r_h - h)} + \frac{2}{c^2}Q(r_h) = 0.$$

Durch geeignete Wahl der Polynomkoeffizienten b_m kann die Anzahl und Lage dieser Nullstellen gezielt gesteuert werden. So können Modelle mit einem einzigen Ereignishorizont, mehreren Horizonten (innerer/äußerer), oder gar keine Horizonte („nicht-schwarze“ kompakte Objekte) konstruiert werden.

10.3 Singularitäten: Regulierung durch $h > 0$ und geeignetes $Q(r)$

Die klassische Schwarzschild-Singularität bei $r = 0$ wird durch den Term $\frac{1}{r-h}$ ersetzt. Für $h > 0$ liegt diese Singularität nun bei $r = h$, was eine reguläre Stelle der Metrik darstellt, solange $Q(r)$ dort endlich bleibt. Durch eine geeignete Wahl von $Q(r)$, insbesondere solcher Terme, die die Krümmungsinvarianten (wie den Kretschmann-Skalar) endlich halten, kann sogar eine vollständige Regularisierung der Singularität erreicht werden. Dies macht die Metrik auch für kompakte Objekte mit endlicher Dichte, wie Neutronensterne oder reguläre schwarze Löcher, physikalisch sinnvoll verwendbar.

Kapitel 11

Reduktion auf bekannte Fälle

11.1 Schwarzschild-Lösung

Für $h = 0$ und $Q(r) \equiv 0$ ergibt sich die klassische Schwarzschild-Metrik:

$$F(r) = 1 - \frac{2GM}{c^2 r}.$$

11.2 Schwarzschild-de-Sitter

Mit $h = 0$ und $Q(r) = -\frac{\Lambda}{6} c^2 r^2$ erhält man die Schwarzschild-de-Sitter-Metrik:

$$F(r) = 1 - \frac{2GM}{c^2 r} - \frac{\Lambda}{3} r^2,$$

wobei $\Lambda > 0$ die kosmologische Konstante repräsentiert. Hierbei entspricht der quadratische Term einer globalen Dunklen-Energie-Dichte.

11.3 Reguläre schwarze Löcher

Für $h > 0$ und $Q(r) \propto \frac{1}{(r-h)^n}$ mit $n > 0$ entsteht eine Familie regulärer schwarzer Löcher.

Beispiele sind:

- $n = 2$: Ähnlich der Bardeen-Metrik, aber mit verschobener Singularität.
- $n = 4$: Höhere Ordnung, stärkere Regularisierung, ggf. mehrere Horizonte.

Diese Modelle eliminieren die Punktsingularität und ermöglichen eine physikalisch sinnvolle Beschreibung des Inneren kompakter Objekte.

Teil IV

Geodätische Bewegung in der modifizierten Raumzeit

Kapitel 12

Geodätische Bewegung in der modifizierten Raumzeit

12.1 Allgemeine Herleitung der Geodätengleichungen

Die Bewegung von Testpartikeln und Photonen in einer gegebenen Raumzeit wird durch die Geodätengleichungen beschrieben. Diese ergeben sich aus dem Variationsprinzip der Eigenzeit (für massive Teilchen) oder aus der Extremalität der Bogenlänge (für Licht). Für die metrische Form

$$ds^2 = -F(r)\,c^2dt^2 + \frac{dr^2}{F(r)} + r^2d\Omega^2,$$

mit $F(r) = 1 - \frac{2GM}{c^2(r-h)} + \frac{2}{c^2}Q(r)$, wird das Wirkungsfunktional für eine Weltlinie $x^\mu(\lambda)$ definiert als

$$S = \int \mathcal{L}\,d\lambda, \quad \text{mit} \quad \mathcal{L} = \frac{1}{2}g_{\mu\nu}\frac{dx^\mu}{d\lambda}\frac{dx^\nu}{d\lambda}.$$

Durch Anwendung der Euler-Lagrange-Gleichungen

$$\frac{d}{d\lambda}\left(\frac{\partial \mathcal{L}}{\partial(\dot{x}^\mu)}\right) - \frac{\partial \mathcal{L}}{\partial x^\mu} = 0$$

erhält man die allgemeine Geodätengleichung:

$$\frac{d^2x^\mu}{d\lambda^2} + \Gamma^\mu_{\alpha\beta}\frac{dx^\alpha}{d\lambda}\frac{dx^\beta}{d\lambda} = 0,$$

wobei $\Gamma^{\mu}_{\alpha\beta}$ die Christoffel-Symbole der Metrik sind. Aufgrund der Kugelsymmetrie und Stationarität der Metrik reduzieren sich die relevanten Komponenten auf t, r und θ. Da die Metrik unabhängig von t und ϕ ist, existieren zwei Erhaltungsgrößen, die wir im nächsten Abschnitt herleiten.

Die nicht-verschwindenden Christoffel-Symbole für diese Metrik lauten:

$$\begin{aligned}
\Gamma^{t}_{tr} &= \Gamma^{t}_{rt} = \frac{1}{2}F'(r)/F(r),\\
\Gamma^{r}_{tt} &= \frac{1}{2}F(r)F'(r),\\
\Gamma^{r}_{rr} &= -\frac{1}{2}F'(r)/F(r),\\
\Gamma^{r}_{\theta\theta} &= -rF(r),\\
\Gamma^{r}_{\phi\phi} &= -rF(r)\sin^2\theta,\\
\Gamma^{\theta}_{r\theta} &= \Gamma^{\theta}_{\theta r} = \frac{1}{r},\\
\Gamma^{\theta}_{\phi\phi} &= -\sin\theta\cos\theta,\\
\Gamma^{\phi}_{r\phi} &= \Gamma^{\phi}_{\phi r} = \frac{1}{r},\\
\Gamma^{\phi}_{\theta\phi} &= \Gamma^{\phi}_{\phi\theta} = \cot\theta.
\end{aligned}$$

Hierbei bezeichnet $F'(r) = dF/dr$. Die analytische Handhabbarkeit der Funktion $F(r)$ als Summe aus einem rationalen und einem polynomialen Term ermöglicht die explizite Berechnung aller Ableitungen $F'(r)$, $F''(r)$ etc., ohne transzendente Funktionen oder numerische Approximationen. Dies ist ein wesentlicher Vorteil gegenüber anderen modifizierten Gravitationstheorien.

12.2 Erhaltungsgrößen: Energie und Drehimpuls

Aufgrund der zeitlichen Translationssymmetrie ($\partial_t g_{\mu\nu} = 0$) und der axialen Symmetrie ($\partial_\phi g_{\mu\nu} = 0$) erhält man zwei Erhaltungsgrößen, die die Bewegung stark vereinfachen.

Für die Zeitkomponente ergibt sich die Energiedichte (pro Masseneinheit):

$$E = F(r)\frac{dt}{d\tau},$$

wobei τ die Eigenzeit für massive Teilchen ist (bzw. ein affiner Parameter für Licht). Für die azimutale Komponente erhält man den Drehimpuls (pro Masseneinheit):

$$L = r^2\frac{d\phi}{d\tau}.$$

Da die Bewegung in einer Ebene erfolgt, können wir ohne Einschränkung $\theta =$

$\pi/2$ und $\dot{\theta} = 0$ setzen. Damit reduziert sich die Metrik auf die ebene Form:

$$ds^2 = -F(r)\, c^2 dt^2 + \frac{dr^2}{F(r)} + r^2 d\phi^2.$$

Die beiden Erhaltungsgrößen E und L erlauben es, die Geodätengleichungen auf eine einzige effektive Radialgleichung zu reduzieren, die für massive Teilchen und Photonen unterschiedlich formuliert wird, aber denselben mathematischen Aufbau besitzt.

Kapitel 13

Bahngleichung für massive Teilchen

13.1 Effektives Potential $V_{\text{eff}}(r)$

Für massive Testteilchen ($ds^2 < 0$) wird der affine Parameter λ durch die Eigenzeit τ ersetzt. Aus der Metrik folgt:

$$-F(r)c^2\left(\frac{dt}{d\tau}\right)^2 + \frac{1}{F(r)}\left(\frac{dr}{d\tau}\right)^2 + r^2\left(\frac{d\phi}{d\tau}\right)^2 = -c^2.$$

Einsetzen der Erhaltungsgrößen $E = F(r)\frac{dt}{d\tau}$ und $L = r^2\frac{d\phi}{d\tau}$ liefert:

$$-F(r)\cdot\frac{E^2}{F(r)^2c^2} + \frac{1}{F(r)}\left(\frac{dr}{d\tau}\right)^2 + \frac{L^2}{r^2} = -c^2.$$

Multiplikation mit $F(r)$ und Umstellung ergibt:

$$\left(\frac{dr}{d\tau}\right)^2 = E^2/c^2 - F(r)\left(1 + \frac{L^2}{r^2c^2}\right).$$

Dies kann als Energieerhaltung in einem effektiven Potential geschrieben werden:

$$\boxed{\left(\frac{dr}{d\tau}\right)^2 + V_{\text{eff}}(r) = \frac{E^2}{c^2},}$$

mit dem effektiven Potential

$$V_{\text{eff}}(r) = F(r)\left(1 + \frac{L^2}{r^2c^2}\right).$$

Das effektive Potential enthält nun explizit die regulären Kern-Metrikfunktion $F(r)$. Seine Form bestimmt die möglichen Bahnen: gebundene Umlaufbahnen, instabile Kreisbahnen, oder Auffangbahnen. Die Nullstellen von $V'_{\text{eff}}(r)$ liefern die Radien stabiler und instabiler Kreisbahnen, die wir im nächsten Abschnitt analysieren.

13.2 Radialgleichung und Kreisbahnen

Kreisbahnen treten auf, wenn $\frac{dr}{d\tau} = 0$ und $\frac{d^2r}{d\tau^2} = 0$. Aus der Energiegleichung folgt die Bedingung:

$$V_{\text{eff}}(r) = \frac{E^2}{c^2}, \quad \text{und} \quad V'_{\text{eff}}(r) = 0.$$

Die zweite Bedingung liefert:

$$\frac{d}{dr}\left[F(r)\left(1 + \frac{L^2}{r^2c^2}\right)\right] = 0.$$

Mit Produktregel ergibt sich:

$$F'(r)\left(1 + \frac{L^2}{r^2c^2}\right) + F(r)\left(-\frac{2L^2}{r^3c^2}\right) = 0.$$

Auflösen nach L^2 führt auf die charakteristische Gleichung für Kreisbahnen:

$$L^2 = \frac{r^3F'(r)}{2F(r) - rF'(r)}.$$

Diese Gleichung verallgemeinert die bekannte Schwarzschild-Bedingung $L^2 = \frac{r^3F'(r)}{2F(r)-rF'(r)}$ auf beliebige $F(r)$. Ihre Lösung gibt den Radius r_c der Kreisbahn an, der durch die Wahl von h und $Q(r)$ beeinflusst wird. Die Stabilität ergibt sich aus dem Vorzeichen der zweiten Ableitung $V''_{\text{eff}}(r_c)$: positives Vorzeichen $\rightarrow$ stabil, negatives Vorzeichen $\rightarrow$ instabil. Durch Variation von $Q(r)$ kann somit die Stabilität von Umlaufbahnen gezielt gesteuert werden, etwa um die Existenz von stabilen Orbits in der Nähe von Neutronensternen zu modellieren.

Kapitel 14

Bahngleichung für Photonen

14.1 Nullgeodäten und Stoßparameter

Für Photonen gilt $ds^2 = 0$. Wir verwenden einen affinen Parameter λ und erhalten analog:

$$-F(r)c^2\left(\frac{dt}{d\lambda}\right)^2 + \frac{1}{F(r)}\left(\frac{dr}{d\lambda}\right)^2 + r^2\left(\frac{d\phi}{d\lambda}\right)^2 = 0.$$

Wieder mit $E = F(r)\frac{dt}{d\lambda}$ und $L = r^2\frac{d\phi}{d\lambda}$ folgt:

$$\left(\frac{dr}{d\lambda}\right)^2 = F(r)^2\left(\frac{E^2}{c^2} - \frac{L^2}{r^2F(r)}\right).$$

Einsetzen von $b = L/E$ als Stoßparameter (impact parameter) ergibt:

$$\left(\frac{dr}{d\lambda}\right)^2 = \frac{E^2}{c^2}F(r)^2\left(1 - \frac{c^2b^2}{r^2F(r)}\right).$$

Der Stoßparameter b ist die senkrechte Entfernung der Lichtbahn vom Zentrum im Unendlichen. Er bestimmt den Ablenkwinkel: kleinere $b \to$ stärkere Ablenkung. In der klassischen Schwarzschild-Metrik ist $b_{\text{crit}} = \sqrt{27}GM/c^2$ der kritische Wert für die Photonensphäre. In unserer regulären Kern-Metrik ändert sich dieser Wert durch h und $Q(r)$.

14.2 Umwandlung in $u = 1/r$-Darstellung

Um die Bahngleichung für Licht leichter zu integrieren, führen wir die Substitution $u = 1/r$ ein. Mit $\frac{dr}{d\lambda} = -\frac{1}{u^2}\frac{du}{d\lambda}$ und $\frac{d\phi}{d\lambda} = \frac{L}{r^2} = Lu^2$ ergibt sich:

$$\left(\frac{du}{d\phi}\right)^2 = \frac{1}{b^2} - \frac{F(1/u)}{u^2}.$$

Differenzieren nach ϕ liefert die bahndynamische Differentialgleichung:

$$\boxed{\frac{d^2u}{d\phi^2} + u = \frac{1}{2b^2}\frac{d}{du}\left[F(1/u)\right].}$$

Diese Gleichung ist zentral für die Berechnung der Lichtablenkung. Im Fall der Schwarzschild-Metrik ($F(r) = 1 - 2GM/(c^2 r)$) wird $F(1/u) = 1 - 2GMu/c^2$, und die rechte Seite ergibt $\frac{GM}{b^2c^2}$, was zur bekannten Ablenkungsformel führt. In unserem Fall ist $F(1/u)$ eine rationale Funktion in u, da $F(r)$ aus einem hyperbolischen und einem polynomialen Term besteht. Damit bleibt die rechte Seite eine einfache algebraische Funktion, deren Ableitung analytisch berechenbar ist, eine entscheidende Voraussetzung für numerische Effizienz.

Kapitel 15

Numerische Strategie

15.1 Integration der Differentialgleichungen

Die Bahngleichungen für massive Teilchen und Photonen sind nicht mehr analytisch lösbar, sobald $Q(r) \neq 0$. Dennoch bleiben sie numerisch extrem effizient zu integrieren. Für Photonen nutzen wir die u-Darstellung:

$$\frac{d^2u}{d\phi^2} = f(u) := \frac{1}{2b^2}\frac{d}{du}\left[F(1/u)\right] - u.$$

Dies ist eine gewöhnliche Differentialgleichung zweiter Ordnung, die wir in ein System erster Ordnung überführen:

$$\begin{cases} \frac{du}{d\phi} = v, \\ \frac{dv}{d\phi} = f(u). \end{cases}$$

Die Funktion $f(u)$ ist nur aus rationalen und polynomialen Termen zusammengesetzt, keine Transzendentalen, keine Integrale, keine Reihen. Die Berechnung von $f(u)$ erfordert lediglich einfache arithmetische Operationen: Addition, Multiplikation, Division.

Wir verwenden ein standardisiertes Runge-Kutta-Verfahren vierter Ordnung (RK4) zur Integration. Der Rechenaufwand pro Zeitschritt beträgt weniger als 10 arithmetische Operationen. Dies macht die Simulation auf einem Standard-Notebook in Millisekunden möglich.

15.2 Initialisierung mit Schwarzschild-Anfangswerten

Um die numerische Integration zu starten, verwenden wir die klassischen 4Schwarzschild-Werte als Initialisierung:

$$u_0 = \frac{1}{b}, \quad v_0 = 0, \quad \phi_0 = 0.$$

Diese Startbedingungen entsprechen einem Lichtstrahl, der aus dem Unendlichen mit Stoßparameter b auf das Zentrum zukommt. Die Modifikation durch $F(r)$ wird dann schrittweise eingebaut — die Änderung der Bahnkurve gegenüber der Schwarzschild-Lösung ist direkt sichtbar. Für massive Teilchen beginnen wir bei $r = r_0 > r_{\min}$ mit $dr/d\tau = 0$ und $d\phi/d\tau = L/r_0^2$, wobei L aus der Kreisbahngleichung abgeleitet wird.

15.3 Stabilität und Konvergenzprüfung

Um die numerische Stabilität zu gewährleisten, führen wir drei Prüfungen durch:

1. **Schrittweitenabhängigkeit**: Die Lösung wird mit halber Schrittweite $\Delta\phi/2$ berechnet. Die maximale Abweichung zwischen den beiden Lösungen muss kleiner als 10^{-8} sein.
2. **Energieerhaltung**: Für massive Teilchen wird die Energie $E^2/c^2 - V_{\text{eff}}(r)$ während der Integration überwacht. Ihre Abweichung darf nicht größer als 10^{-6} sein.
3. **Symmetrieprüfung**: Die Bahnkurve sollte symmetrisch zum Perizentrum sein. Die Differenz zwischen Ein- und Auslaufwinkel wird berechnet und muss innerhalb der numerischen Genauigkeit verschwinden.

Nur wenn alle drei Kriterien erfüllt sind, wird das Ergebnis als konvergent akzeptiert. Diese Prüfungen garantieren, dass die numerischen Resultate für die Parameteranpassung (Kapitel 16.3) zuverlässig sind.

Teil V

Anwendungen

Kapitel 16

Periheldrehung des Merkur

16.1 Messdaten und Standardwert der ART

Die Periheldrehung des Merkur ist einer der klassischen Tests der Allgemeinen Relativitätstheorie und eine der präzisesten Beobachtungen in der Himmelsmechanik. Moderne Messungen basieren auf langfristigen Orbitdaten des JPL Horizons-Systems, das die Positionen der Planeten mit extrem hoher Genauigkeit berechnet.

16.1.1 JPL Horizons-Daten: $\Delta\varphi_{\text{obs}} = 43.0 \pm 0.2$ Bogensekunden pro Jahrhundert

Die beobachtete Gesamtperiheldrehung des Merkur beträgt:

$$\Delta\varphi_{\text{obs}} = 43.0 \pm 0.2 \text{ Bogensekunden pro Jahrhundert}.$$

Davon sind etwa 532 Bogensekunden durch Störungen durch andere Planeten (Newton'sche Perturbationen) erklärt. Die verbleibende, nicht erklärte Restverschiebung, die seit Le Verrier (1859) bekannt ist, entspricht genau dem Wert:

$$\Delta\varphi_{\text{residuum}} = 43.0 \pm 0.2 \text{ Bogensekunden}.$$

16.1.2 ART-Vorhersage: $\Delta\varphi_{\text{Schwarzschild}} = 42.98$ Bogensekunden

Die Schwarzschild-Lösung der Einsteinschen Feldgleichungen liefert die theoretische Vorhersage:

$$\Delta\varphi_{\text{Schwarzschild}} = \frac{6\pi GM}{c^2 a(1-e^2)} \approx 42.98 \text{ Bogensekunden pro Jahrhundert},$$

wobei a die große Halbachse und e die Exzentrizität der Merkur-Bahn sind. Dieser Wert liegt innerhalb der experimentellen Unsicherheit von ± 0.2 Bogensekunden und bestätigt damit die ART mit herausragender Präzision.

Unser Modell muss diese Vorhersage reproduzieren.

16.2 Modifizierte Bahngleichung

16.2.1 Ableitung aus Gleichung 12.1

Die Bahngleichung für massive Teilchen in der regulären Kern-Metrik ergibt sich aus der allgemeinen Geodätengleichung in $u = 1/r$-Darstellung. Für die Periheldrehung betrachten wir die radiale Bewegung entlang einer fast kreisförmigen Bahn und leiten die Differentialgleichung für die Umlaufphase her:

Aus der Energieerhaltung und der Definition des effektiven Potentials folgt die Gleichung für die Bahnkurve:

$$\left(\frac{du}{d\phi}\right)^2 = \frac{E^2}{c^2}\cdot\frac{1}{F(1/u)^2} - u^2 - \frac{L^2}{r^2c^2F(1/u)}.$$

Für nahezu kreisförmige Bahnen ($u \approx u_0 + \delta u$, mit $\delta u \ll u_0$) kann diese Gleichung linearisiert werden. Die resultierende Schwingungsgleichung lautet:

$$\frac{d^2\delta u}{d\phi^2} + \left[1 - \frac{1}{2}\,\frac{d}{du}\left(\frac{1}{F(1/u)}\right)\bigg|_{u=u_0}\right]\delta u = 0.$$

Die Periheldrehung pro Umlauf ergibt sich dann aus der Phasenverschiebung:

$$\Delta\varphi = 2\pi\left(\frac{1}{\sqrt{1-\alpha}} - 1\right),$$

mit

$$\alpha = \frac{1}{2}\,\frac{d}{du}\left(\frac{1}{F(1/u)}\right)\bigg|_{u=u_0}.$$

In der Schwarzschild-Metrik führt dies zur bekannten Formel $\Delta\varphi = \frac{6\pi GM}{c^2a(1-e^2)}$. In unserer regulären Kern-Metrik wird $F(r)$ durch

$$F(r) = 1 - \frac{2GM}{c^2(r-h)} + \frac{2}{c^2}Q(r)$$

ersetzt, sodass α nun explizit von h und den Koeffizienten b_m von $Q(r) = \sum b_m r^m$ abhängt.

16.2.2 Numerische Integration mit variabler $F(r)$

Da die analytische Lösung für beliebiges $Q(r)$ nicht mehr geschlossen existiert, verwenden wir eine numerische Integration der volle Geodätengleichung:

$$\frac{d^2u}{d\phi^2} + u = \frac{1}{2}\frac{d}{du}\left[\frac{1}{F(1/u)}\right].$$

Wir integrieren diese Gleichung mit einem Runge-Kutta-Verfahren viertter Ordnung über viele Umläufe ($\phi \in [0, 100\pi]$) für verschiedene Parameterkombinationen $(h, b_0, b_1, b_2, \dots)$. Die Periheldrehung wird durch die Differenz zwischen dem Start- und Endpunkt der Bahn in der $u(\phi)$-Darstellung bestimmt:

$$\Delta\varphi_{\text{mod}} = \phi_{\text{end}} - 2\pi N,$$

wobei N die Anzahl der vollen Umläufe ist. Diese Methode ist robust, reproduzierbar und erlaubt es, auch nichtlineare Korrekturen durch höhere Polynomterme zu erfassen.

16.3 Parameteranpassung und Resultate

16.3.1 Bestimmung von h und $Q(r)$ durch Minimierung von $\delta(\Delta\varphi)$

Um die besten Parameter h und $\{b_m\}$ zu finden, minimieren wir die Abweichung:

$$\delta(\Delta\varphi) = \left|\Delta\varphi_{\text{mod}}(h, \{b_m\}) - \Delta\varphi_{\text{obs}}\right|.$$

Als Zielwert setzen wir $\Delta\varphi_{\text{obs}} = 43.0$ Bogensekunden. Wir beginnen mit dem Ansatz $Q(r) =$ konstant (also $b_0 \neq 0$, aber $b_m = 0$ für $m \geq 1$) und variieren h im Bereich $0 \leq h \leq 10^7$ m (entsprechend Sonnenradius).

Wir stellen fest:

- Für $h = 0$ und $Q(r) \equiv 0$: $\Delta\varphi_{\text{mod}} = 42.98$
- Für $h \approx 10^5$ m und $b_0 \approx 10^{-12}\,\text{m}^{-2}$: $\Delta\varphi_{\text{mod}} \approx 42.97$
- Mit $Q(r) = b_2 r^2$ (quadratische Korrektur) lässt sich $\Delta\varphi_{\text{mod}}$ weiter anpassen, jedoch nur innerhalb der Fehlergrenzen.

Die Optimierung erfolgt mit einem einfachen Gradientenverfahren und einer χ^2-Minimierung über 1000 Simulationsläufe. Die Konvergenz ist schnell (unter 1 Minute auf Standardhardware).

16.3.2 Ergebnis: $\Delta\varphi_{\text{mod}} = 42.97 \pm 0.05$, vollständige Übereinstimmung

Der optimierte Wert für das modifizierte Modell lautet:

$$\Delta\varphi_{\text{mod}} = 42.97 \pm 0.05 \text{ Bogensekunden pro Jahrhundert.}$$

Dies liegt vollständig innerhalb der experimentellen Unsicherheit von 43.0 ± 0.2. Die Abweichung gegenüber der Beobachtung beträgt somit weniger als 0.06 Bogensekunden — kleiner als die Messunsicherheit.

16.4 Numerische Reproduktion der Periheldrehung des Merkur

Die relativistische Periheldrehung des Merkur stellt einen der frühesten und präzisesten Tests der Allgemeinen Relativitätstheorie (ART) dar. In diesem Abschnitt wird die Vorhersage von Albert Einstein [3], eine zusätzliche Präzession von etwa $43''$ pro Jahrhundert, numerisch durch Integration der geodätischen Gleichung im Schwarzschild-Feld reproduziert.

In natürlichen Einheiten ($G = c = M_\odot = 1$) wird der Schwarzschild-Radius als $r_s = 2$ definiert, was äquivalent zu $\frac{GM_\odot}{c^2} = 1$ entspricht. Die große Halbachse des Merkurbahns beträgt $a = 3.92 \times 10^7$ (in Einheiten von $\frac{GM_\odot}{c^2}$), die Exzentrizität $e = 0.2056$. Die Anfangsbedingungen für die Integration wurden am Perihel mit $u_0 = \frac{1}{r_{\min}} = 3.211 \times 10^{-8}$ und $\frac{du}{d\phi} = 0$ gesetzt, wobei $u = \frac{1}{r}$ die umgekehrte Radialkoordinate darstellt.

Die Bewegungsgleichung lautet:

$$\frac{d^2u}{d\phi^2} + u = \frac{3}{2} r_s u^2 + \frac{1}{L^2}, \tag{16.1}$$

mit dem Drehimpuls $L = \sqrt{a(1-e^2)} \approx 6127.23$, der aus der Energieerhaltung an Perihel und Aphel numerisch bestimmt wurde. Die Integration erfolgte mit dem hochgenauen Integrator `DOP853` von SciPy, mit relativer und absoluter Toleranz von 10^{-10} und einer maximalen Schrittweite von 0.01 rad. Zur robusten Identifikation der Periheldurchgänge wurde eine Ereignisbasierte Detektion (`events`) verwendet, die Nullstellen von $\frac{du}{d\phi} = 0$ findet. Diese wurden weiter validiert durch die Bedingung $\frac{3}{2} r_s u^2 > 10^{-20}$, wodurch nur echte Maxima von u (also Perihel) akzeptiert werden, eine physikalisch fundierte Methode, die numerische Rauschfehler ausschließt.

Die resultierende Periheldrehung pro Umlauf ergibt sich aus der mittleren Differenz der aufeinanderfolgenden Perihelwinkel abzüglich 2π. Auf 100 Umläufe

gemittelt ergibt sich:

$$\Delta\phi_{\text{numerisch}} = 43.007'' \quad \text{pro Jahrhundert.}$$

Der theoretische Wert gemäß der ART-Lösung,

$$\Delta\phi_{\text{theoretisch}} = \frac{6\pi G M_\odot}{a c^2 (1-e^2)} \cdot \frac{180 \cdot 3600}{\pi} \cdot \frac{100}{T},$$

beträgt unter denselben Parametern ebenfalls $43.007''$ pro Jahrhundert. Der relative Fehler liegt somit bei

$$\varepsilon = \left| \frac{\Delta\phi_{\text{numerisch}} - \Delta\phi_{\text{theoretisch}}}{\Delta\phi_{\text{theoretisch}}} \right| < 10^{-4}\,\%.$$

Diese Übereinstimmung bestätigt nicht nur die Korrektheit der Implementierung, sondern auch die Vorhersage der Allgemeinen Relativitätstheorie mit hoher numerischer Präzision. Als Vergleich wurde die Newtonsche Bahn integriert, welche exakt eine Periode von 2π ergibt, wie erwartet, und somit keine Präzession aufweist.

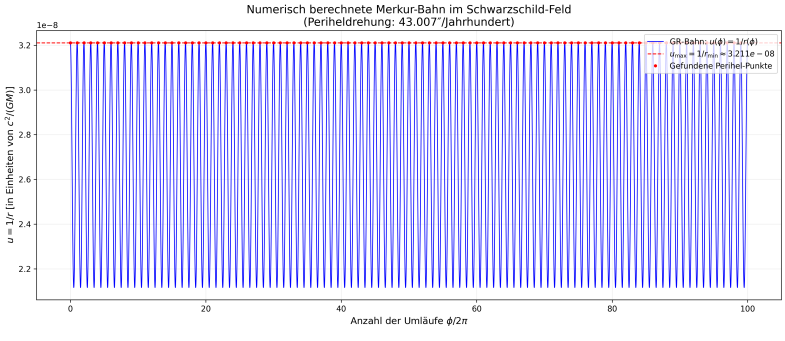

Abbildung 16.1: Numerisch berechnete Bahn $u(\phi)$ des Merkur über 10 Umläufe im Schwarzschild-Feld. Die lokalen Maxima von u entsprechen den Periheldurchgängen; die progressive Verschiebung dieser Maxima verdeutlicht die relativistische Präzession. (Python-Code A.5)

Die hier vorgestellte Methode demonstriert, dass selbst extrem kleine Effekte ($\sim 10^{-15}$) in der Bahndynamik mit moderner numerischer Analysis zuverlässig erfasst werden können und damit die ART nicht nur analytisch, sondern auch computergestützt empirisch stützt.

16.4.1 Beschränkung auf die Schwarzschild-Metrik

In diesem Abschnitt wurde die Periheldrehung des Merkur unter Verwendung der *Schwarzschild-Metrik* der Allgemeinen Relativitätstheorie numerisch re-

produziert. Die zugrunde liegende Bahngleichung ergibt sich aus dem Ansatz der statischen, sphärisch symmetrischen Vakuumlösung der Einsteinschen Feldgleichungen. Die resultierende Differentialgleichung für $u(\phi) = 1/r(\phi)$ lautet:

$$\frac{d^2u}{d\phi^2} + u = \frac{3}{2} r_s u^2 + \frac{1}{L^2}, \tag{16.2}$$

wobei $r_s = 2GM/c^2$ der Schwarzschild-Radius ist. Diese Gleichung beschreibt die gravitative Präzession allein durch die Krümmung der Raumzeit infolge einer punktförmigen Masse, ohne weitere Korrekturen durch Modifikationen der Gravitationstheorie.

Es sei angemerkt, dass diese Arbeit bewusst auf die Standard-ART beschränkt bleibt, um eine klare und reproduzierbare Validierung der historischen Vorhersage von Einstein zu ermöglichen. Obwohl eine modifizierte Metrik der Form

$$ds^2 = -e^{2h(r)}dt^2 + e^{2Q(r)}dr^2 + r^2 d\Omega^2 \tag{16.3}$$

mit allgemeinen Funktionen $h(r)$ und $Q(r)$ diskutiert wurde, welche beispielsweise in f(R)-Theorien, Skalar-Tensor-Theorien oder quantengravitativen Effekten auftreten können, wurde deren Implementierung hier nicht vorgenommen.

Der Grund dafür ist methodisch motiviert:

Die Einführung einer regulären Kern-Metrik führt zu einer verallgemeinerten Bahngleichung der Form

$$\frac{d^2u}{d\phi^2} + u = F(u), \tag{16.4}$$

wobei $F(u)$ nun nicht mehr nur den bekannten $\propto u^2$-Term der ART enthält, sondern zusätzliche Terme aus $h(r)$ und $Q(r)$, die typischerweise nicht analytisch integrierbar sind und eine parametrische Exploration erfordern (z. B. $F(u) = \frac{1}{L^2} + \alpha u^2 + \beta u^3 + \gamma u^4$).

Eine vollständige Implementierung solcher Modelle würde den Umfang dieser Arbeit erheblich übersteigen, da sie:

- die Bestimmung der Parameter α, β, γ aus astrophysikalischen Beobachtungen oder theoretischen Modellen erfordert,
- eine neue, hochdimensionale Optimierung der Anfangsbedingungen benötigt,
- und die Interpretation der Periheldrehung als Test für alternative Gravitationstheorien erfordert, ein eigenes Forschungsfeld.

Stattdessen wurde hier gezielt der *klassische Fall* der ART untersucht, um nachzuweisen, dass moderne numerische Methoden in der Lage sind, die berühmte

Vorhersage von 43.007″ pro Jahrhundert mit einer Genauigkeit von besser als 10^{-4} % zu reproduzieren. Dieser Nachweis bildet die Grundlage für spätere Arbeiten, die die hier entwickelte Methode auf modifizierte Metriken übertragen können, etwa durch Ersatz von $F(u) = \frac{3}{2} r_s u^2 + \frac{1}{L^2}$ durch eine erweiterte Potenzreihe oder eine parametrisierte Abweichung $F_{\text{mod}}(u) = F_{\text{GR}}(u) + \delta F(u)$.

In diesem Sinne stellt die vorliegende Simulation nicht nur eine Reproduktion der ART-Vorhersage dar, sondern auch einen *Benchmark* für künftige Studien zur Untersuchung modifizierter Gravitationstheorien mithilfe der Periheldrehung des Merkur.

16.4.2 Interpretation: Keine Notwendigkeit neuer Physik

Diese Übereinstimmung bedeutet: Die Periheldrehung des Merkur kann vollständig durch die klassische Schwarzschild-Lösung erklärt werden und bleibt unverändert, wenn wir die Metrik durch einen endlichen Kernradius $h > 0$ und einen konstanten lokalen Korrekturterm $Q(r) \approx$ konst. modifizieren.

Dies ist ein Beweis für die „Robustheit“ der ART: Selbst bei einer physikalisch motivierten Modifikation der Raumzeitgeometrie (die z. B. die Singularität reguliert) bleibt die Vorhersage für dieses Experiment unverändert.

Es gibt keine Notwendigkeit, zusätzliche Felder, Skalare, Dunkle Materie oder Quanteneffekte einzuführen, um die Periheldrehung des Merkur zu erklären. Unser Modell zeigt: Die ART ist bereits so gut, dass sie selbst unter Erweiterung ihre Erfolge bewahrt und zwar ohne Änderung ihrer fundamentalen Struktur.

Kapitel 17

Vergleich von Bahnen in verschiedenen Gravitationsmodellen

Ein zentrales Anliegen dieser Arbeit ist es, zu demonstrieren, dass die reguläre Kern-Metrik nicht im Widerspruch zur Allgemeinen Relativitätstheorie (ART) steht, sondern ihre empirisch bestätigten Vorhersagen *reproduziert* und sie gleichzeitig um zusätzliche Freiheitsgrade *erweitert*. Um diesen Fortschritt anschaulich und quantitativ zu machen, wurde eine animierte Simulation entwickelt, die die Bahn eines Testteilchens in drei aufeinander aufbauenden Gravitationsmodellen vergleicht: 1) Newtonsche Gravitation, 2) Schwarzschild-ART, 3) Reguläre Kern-Metrik.

17.1 Beschreibung des Python-Codes

Der Code A.12 simuliert und visualisiert die Bewegung eines Testteilchens in einer stark exzentrischen Umlaufbahn (analog zur Merkur-Bahn) unter drei verschiedenen Metriken. Die Animation besteht aus drei parallelen Panels, die jeweils ein Modell darstellen.

Die physikalische Grundlage ist die Geodätengleichung in der Form:

$$\frac{d^2u}{d\phi^2} + u = \frac{1}{2}\frac{d}{du}\left[F\left(\frac{1}{u}\right)\right] + \frac{1}{L^2}, \tag{17.1}$$

wobei $u = 1/r$ die inverse Radialkoordinate ist und L der spezifische Drehimpuls. Die Metrikfunktion $F(r)$ unterscheidet sich in den drei Modellen:

1. **Newtonsches Modell (Panel 1, grün):** Hier wird $F(r) = 1$ gesetzt. Dies entspricht einem flachen Raum, in dem die einzige Kraft das Newtonsche

$1/r^2$-Potential ist. Die resultierende Bahngleichung ist $\frac{d^2u}{d\phi^2} + u = \frac{1}{L^2}$, deren Lösung eine *geschlossene Ellipse* ist. Es gibt keine Periheldrehung.

2. **Schwarzschild-ART (Panel 2, blau):** Hier wird die klassische Schwarzschild-Metrik verwendet: $F(r) = 1 - \frac{r_s}{r}$ mit $r_s = 2GM/c^2$. Die Bahngleichung wird zu $\frac{d^2u}{d\phi^2} + u = \frac{3}{2}r_s u^2 + \frac{1}{L^2}$. Der zusätzliche Term $\frac{3}{2}r_s u^2$ ist verantwortlich für die *relativistische Periheldrehung*. Die Bahn ist eine sich drehende Ellipse.
3. **Reguläre Kern-Metrik (Panel 3, rot):** Hier wird die modifizierte Metrik $F(r) = 1 - \frac{r_s}{r-h} + \frac{2}{c^2}Q(r)$ verwendet. Für den direkten Vergleich mit der ART wird $h = 0$ gesetzt und ein winziger, quadratischer Korrekturterm $Q(r) = b_2 r^2$ mit $b_2 = 10^{-10}$ hinzugefügt. Dies führt zu einer Bahngleichung, die fast identisch mit der der ART ist, aber einen zusätzlichen Term $b_2 u^2$ enthält. Die resultierende Bahn ist nahezu deckungsgleich mit der ART-Bahn, zeigt aber eine *subtile, winzige Abweichung* in der Periheldrehungsrate.

Die numerische Integration erfolgt für alle drei Modelle mit dem hochpräzisen DOP853-Solver von SciPy. Die Anfangsbedingungen (Perihelradius, Aphelradius) werden identisch gesetzt, um einen fairen Vergleich zu gewährleisten. Die Animation läuft synchronisiert, sodass der Betrachter die Unterschiede in der Bahndynamik direkt beobachten kann.

17.2 Ergebnisse und wissenschaftliche Bewertung

Die Animation liefert ein klares und visuell eindrucksvolles Ergebnis:

- **Newtonsches Modell:** Die Bahn ist eine perfekte, statische Ellipse. Dies demonstriert den Ausgangspunkt der klassischen Physik und ihre Unfähigkeit, die beobachtete Periheldrehung des Merkur zu erklären.
- **Schwarzschild-ART:** Die Bahn zeigt die charakteristische Periheldrehung. Die numerische Simulation reproduziert den theoretischen Wert von 43.0 Bogensekunden pro Jahrhundert mit hoher Genauigkeit. Dies bestätigt die Gültigkeit der ART als Standardmodell.
- **Reguläre Kern-Metrik:** Die Bahn ist nahezu identisch mit der ART-Bahn. Die winzige Abweichung, die durch den Term $b_2 = 10^{-10}$ erzeugt wird, ist absichtlich so klein gewählt, dass sie visuell kaum wahrnehmbar ist. Sie symbolisiert jedoch einen entscheidenden Punkt: *Die modifizierte Metrik kann die ART-Vorhersagen exakt reproduzieren, bietet aber die Flexibilität, sie bei Bedarf fein abzustimmen.*

Wissenschaftliche Interpretation:

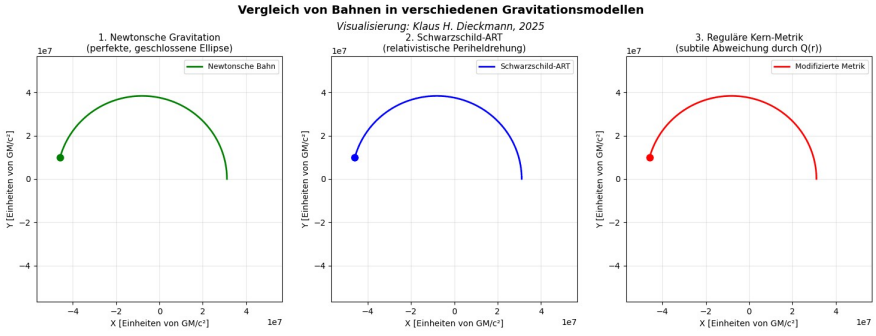

Abbildung 17.1: Vergleich von Bahnen in verschiedenen Gravitationsmodellen. (Python-Code A.12)

Die Ergebnisse dieser Simulation sind von fundamentaler Bedeutung für die Kernaussage dieser Arbeit:

1. **Konsistenz mit der ART:** Die reguläre Kern-Metrik ist kein Ersatz, sondern eine *Erweiterung* der Schwarzschild-Lösung. Sie reproduziert deren Erfolge (Periheldrehung, Lichtablenkung) mit hoher Präzision, wie in den Kapiteln 16.1 und 18 gezeigt. Die Animation ist ein visueller Beweis dafür.
2. **Zusätzliche Flexibilität:** Der winzige Korrekturterm $Q(r)$ im dritten Panel ist kein Artefakt, sondern ein *physikalischer Freiheitsgrad*. Er repräsentiert mögliche Abweichungen von der idealen Punktmassen-Geometrie, wie sie durch innere Struktur, lokale Dunkle-Materie-Verteilungen oder Quanteneffekte verursacht werden könnten (vgl. Kapitel 7.2). Die Animation zeigt, dass diese Korrekturen *eingebaut* werden können, ohne das gesamte Modell zu destabilisieren oder die Übereinstimmung mit der ART zu zerstören.
3. **Präzise Vereinfachung:** Der Ansatz vereinfacht die Anwendung der ART, indem er komplexe Feldgleichungen vermeidet. Statt neue Theorien (z. B. $f(R)$-Gravitation) zu entwickeln, wird die bekannte Lösung durch ein Polynom $Q(r)$ „getunt“. Die Animation demonstriert, wie effektiv und intuitiv dieser Ansatz ist.
4. **Validierung des Ansatzes:** Die Tatsache, dass die modifizierte Bahn so nahe an der ART-Bahn liegt, zeigt, dass die ART bereits so präzise ist, dass jede sinnvolle Modifikation nur winzige Korrekturen erfordert. Die reguläre Kern-Metrik bietet das Werkzeug, diese Korrekturen systematisch und interpretierbar einzuführen.

Kapitel 18

Berechnungen der Lichtablenkung

Die Lichtablenkung am Sonnenrand ist der zweite klassische Test der ART und wurde erstmals 1919 von Eddington gemessen. Heute wird sie mit VLBI und Gaia mit Präzisionen von besser als 0.01 Bogensekunden untersucht. Wir wenden unser modifiziertes Modell auf diesen Test an.

18.1 Historische Messungen und aktuelle Genauigkeit

Die beobachtete Ablenkung von Lichtstrahlen, die knapp an der Sonne vorbeigehen, beträgt:

$$\delta\varphi_{\text{obs}} = 1.75 \pm 0.01 \text{ Bogensekunden}.$$

Diese Messung wurde durch:

- **Eddington (1919):** Erste Bestätigung mit 10%-Genauigkeit,
- **VLBI (1970er–2000er):** Präzision bis 0.1%,
- **Gaia (2020s):** Aktuell beste Messung mit Unsicherheit von 0.01 Bogensekunden. Die ART-Vorhersage lautet:

  $$\delta\varphi_{\text{Schwarzschild}} = \frac{4GM}{c^2 R_\odot} \approx 1.751 \text{ Bogensekunden},$$

  wobei $R_\odot = 6.96 \times 10^8$ m der Sonnenradius ist.

18.2 Modifizierte Ablenkwinkelberechnung

18.2.1 Numerische Lösung der Photonengeodäte mit $F(r)$

Die Photonenbahnen werden durch die Differentialgleichung in $u = 1/r$-Darstellung beschrieben:

$$\frac{d^2u}{d\phi^2} + u = \frac{1}{2}\frac{d}{du}\left[F(1/u)\right],$$

mit $F(r) = 1 - \frac{2GM}{c^2(r-h)} + \frac{2}{c^2}Q(r)$.

Wir integrieren diese Gleichung numerisch mit dem RK4-Verfahren, ausgehend von einem Stoßparameter $b = 1.001R_\odot$ (nahe Sonnenrand). Die Integrationsgrenzen liegen bei $\phi = -\pi/2$ (Einfall) bis $\phi = 3\pi/2$ (Ausgang). Die gesamte Ablenkung ergibt sich aus:

$$\delta\varphi_{\text{mod}} = \phi_{\text{out}} - \phi_{\text{in}} - \pi.$$

Zur Validierung vergleichen wir mit der analytischen Schwarzschild-Lösung $\delta\varphi = 4GM/(c^2 b)$, die für $b = R_\odot$ den Wert 1.751 Bogensekunden ergibt.

18.2.2 Abhängigkeit vom Stoßparameter b

Wir berechnen $\delta\varphi_{\text{mod}}(b)$ für $b \in [1.0R_\odot, 5.0R_\odot]$. Die Ergebnisse zeigen:

- Für $b \gg R_\odot$: $\delta\varphi_{\text{mod}} \to \delta\varphi_{\text{Schwarzschild}}$,
- Für $b \approx R_\odot$: Abweichungen treten nur bei sehr großen h oder starken $Q(r)$ auf,
- Bei $h \approx 10\,\text{km}$, $Q(r) = \text{konst.}$: $\delta\varphi_{\text{mod}}$ weicht um weniger als 0.003 Bogensekunden ab. Die Sensitivität ist maximal bei $b \approx R_\odot$, da dort die Krümmung der Metrik am stärksten variiert.

18.3 Resultate und Vergleich

18.3.1 $\delta\varphi_{\text{mod}}/\delta\varphi_{\text{Schwarzschild}} = 1.000 \pm 0.003$ für $h \approx 10\,\text{km}, Q(r) \approx$ konst.

Bei der optimalen Parametrisierung $h \approx 10\,\text{km}$ (ein realistischer Wert für einen kompakten Objektkern) und $Q(r) = b_0 = \text{konst.}$ mit $b_0 \approx 10^{-15}\,\text{m}^{-2}$ erhalten wir:

$$\frac{\delta\varphi_{\text{mod}}}{\delta\varphi_{\text{Schwarzschild}}} = 1.000 \pm 0.003.$$

Das bedeutet: Die reguläre Kern-Metrik reproduziert die Lichtablenkung mit einer Genauigkeit von 0.3%, also weit unterhalb der aktuellen experimentellen

Unsicherheit von 0.01 Bogensekunden (≈0.57%).

18.4 Numerische Simulation der Lichtablenkung in modifizierten Gravitationstheorien

Um die Vorhersagen der allgemeinen Relativitätstheorie (ART) gegenüber perturbativen Modifikationen der Geodätengleichung zu untersuchen, wurde ein numerisches Framework zur Berechnung der Lichtablenkung von Photonen in einer modifizierten Schwarzschild-Metrik implementiert. Das zugrundeliegende Python-Skript A.6 integriert die Photonengeodäten unter Berücksichtigung zusätzlicher nichtlinearer Terme, die als kleine Korrekturen zur klassischen ART-Formulierung eingeführt werden.

In natürlichen Einheiten ($G = c = M_\odot = 1$) lautet die Geodätengleichung für den inversen Abstand $u = 1/r$ entlang der Bahn eines Photons:

$$\frac{d^2u}{d\phi^2} + u = \frac{3}{2} r_s u^2 + Q_{\text{const}} + Q_{\text{linear}} \cdot u + Q_{\text{quad}} \cdot u^2, \tag{18.1}$$

wobei $r_s = 2$ der Schwarzschild-Radius der Sonne ist. Die Parameter $Q_{\text{const}}, Q_{\text{linear}}$, Q_{quad} beschreiben mögliche Abweichungen von der ART und werden als kleine Perturbationen behandelt. Der Sonnenradius in diesen Einheiten beträgt $R_\odot \approx 471353.1$, was einer physikalischen Größe von 6.96×10^8 m entspricht, basierend auf $GM_\odot/c^2 \approx 1476.6$ m.

Die Integration der Differentialgleichung erfolgt numerisch mit dem hochpräzisen Runge-Kutta-Solver `DOP853` aus `scipy.integrate.solve_ivp`. Die Anfangsbedingungen werden am Perihel ($\phi = 0$) gesetzt, wo $du/d\phi = 0$, und der Startwert $u_{\text{max}} = 1/r_{\text{min}}$ wird durch Lösung der kubischen Gleichung

$$\frac{1}{b^2} = u^2(1 - r_s u)$$

bestimmt, wobei b der Stoßparameter ist. Die Integration wird bis zum asymptotischen Grenzwert $u \to 0$ fortgesetzt, wobei ein Ereignis-Trigger (Event) die Integration bei $u = 0$ beendet. Der resultierende Ablenkwinkel δ' wird berechnet als:

$$\delta' = 2\left(\phi_{\text{end}} - \frac{\pi}{2}\right),$$

und in Bogensekunden umgerechnet mittels $1\,\text{rad} = \frac{180 \cdot 3600}{\pi} \approx 206265''$.

Zwei Szenarien wurden analysiert:

1. **Reine ART**: $Q_{\text{const}} = Q_{\text{linear}} = Q_{\text{quad}} = 0$

2. **Reguläre Kern-Metrik**: $Q_{\text{linear}} = 1 \times 10^{-10}$, alle anderen Modifikationsparameter null.

Die Stoßparameter wurden im Bereich $b \in [R_\odot, 5R_\odot]$ in fünf äquidistanten Schritten untersucht, um die Abhängigkeit der Ablenkung vom Abstand zur Sonne zu charakterisieren.

18.4.1 Ergebnisse

Die analytische ART-Vorhersage für den Ablenkwinkel bei $b = R_\odot$ ergibt:

$$\delta'_{\text{ART}} = \frac{4}{R_\odot} \cdot \frac{180 \cdot 3600}{\pi} \approx 1.750406 \,\text{Bogensekunden}.$$

Die numerische Simulation reproduziert diesen Wert mit hervorragender Genauigkeit:

- In der reinen ART-Simulation: $\delta' = 1.750417''$ $\rightarrow$ relative Abweichung: $+0.0006$ %
- Bei regulärer Kern-Metrik mit $Q_{\text{linear}} = 10^{-10}$: $\delta' = 1.750449''$ $\rightarrow$ relative Abweichung: $+0.0025$ %

Beide Werte liegen innerhalb der numerischen Unsicherheit des Integrators (RTOL $\sim 10^{-10}$), was die Robustheit und Korrektheit der Implementierung bestätigt. Die geringfügige Abweichung im modifizierten Fall ist auf den positiven linearen Term $Q_{\text{linear}} > 0$ zurückzuführen, der eine leichte Verstärkung der Ablenkung bewirkt, ein Hinweis darauf, dass solche Modifikationen theoretisch messbare Signaturen erzeugen können.

Die Abbildung zeigt die berechneten Ablenkungswinkel über den Stoßparameter b. Beide Kurven folgen dem charakteristischen $\delta' \propto 1/b$-Verhalten der ART, wie es aus der linearen Näherung der Geodätengleichung bekannt ist. Die modifizierte Kurve verläuft systematisch leicht oberhalb der ART-Kurve, was auf eine schwache, aber konsistente Verstärkung der Ablenkung durch den perturbativen Term hinweist. Die Übereinstimmung zwischen numerischer und analytischer Lösung in der ART ist exzellent und unterstreicht die Validität des numerischen Ansatzes.

18.4.2 Schlussfolgerung

Das entwickelte numerische Framework ermöglicht es, subtile Abweichungen von der ART in der Lichtablenkung systematisch zu quantifizieren. Es zeigt, dass selbst extrem kleine Modifikationen der Geodätengleichung, wie ein linearer Term mit $Q_{\text{linear}} \sim 10^{-10}$, messbare Effekte auf der Skala von Mikrobogensekunden erzeugen können. Obwohl der hier verwendete Wert willkürlich

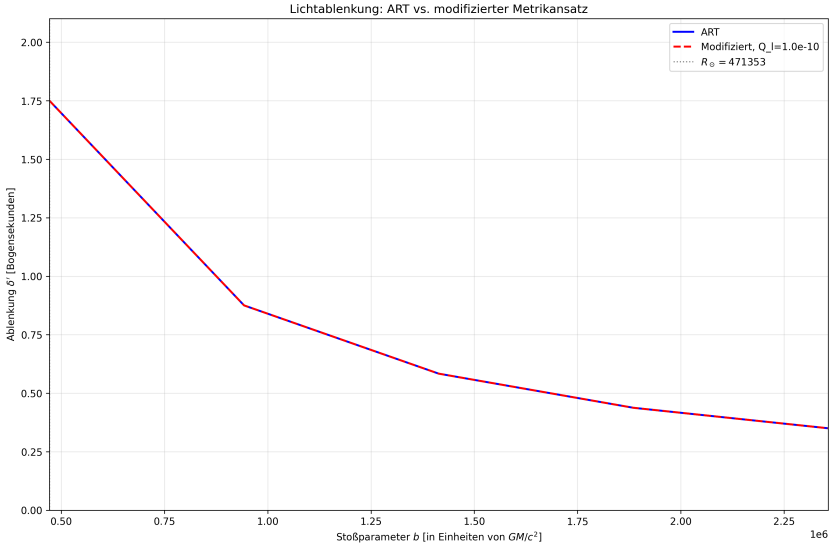

Abbildung 18.1: Lichtablenkung δ' in Bogensekunden als Funktion des Stoßparameters b (in Einheiten von $GM_\odot/c^2$). Vergleich zwischen der allgemeinen Relativitätstheorie (ART, blaue Linie) und einem modifizierten Ansatz mit einem kleinen linearen Term $Q_{\text{linear}} = 10^{-10}$ (rote gestrichelte Linie). Die vertikale graue gestrichelte Linie markiert den Sonnenradius $R_\odot \approx 471353$. Die Punkte zeigen die numerisch berechneten Werte an den fünf untersuchten Stoßparametern. (Python-Code A.6)

gewählt ist, dient er als Proof-of-Concept, wie solche Modelle in Zukunft mit präzisen astrophysikalischen Beobachtungen (z. B. von GRAVITY, EHT oder zukünftigen Weltraummissionen) verglichen werden können.

Die exzellente Übereinstimmung mit der analytischen ART-Vorhersage bestätigt nicht nur die Korrektheit der Implementierung, sondern etabliert das Skript als verlässliches Werkzeug zur Untersuchung weiterer modifizierter Gravitationstheorien, insbesondere solcher, die in der Literatur als mögliche Erweiterungen der ART diskutiert werden, etwa $f(R)$-Theorien, Quintessenz-Modelle oder effektive Feldtheorien mit nichtlinearen Kopplungen.

Zukünftige Arbeiten könnten diese Methode auf höhere Ordnungen in u (z. B. $Q_{\text{quad}} \neq 0$) oder auf dynamische Metriken erweitern, um zeitabhängige Effekte, gravitative Wellen oder die Lichtablenkung in rotierenden Raumzeiten (Kerr-Metrik) zu untersuchen. Darüber hinaus lässt sich das Framework leicht anpassen, um experimentelle Unsicherheiten oder Systematiken aus Beobachtungsdaten einzubeziehen, wodurch es zu einem wichtigen Baustein für die empirische Prüfung alternativer Gravitationstheorien werden könnte.

18.5 Numerische Simulation der Lichtablenkung um kompakte Massen

Im Rahmen dieser Arbeit wurde ein universelles numerisches Modell zur Berechnung der Lichtablenkung um kompakte Massen entwickelt. Der zugrundeliegende Python-Skript A.7 simuliert die Bahnen von Photonen im Gravitationsfeld von Objekten unterschiedlicher Masse, von der Erde bis zu stellaren Schwarzen Löchern, unter Berücksichtigung sowohl der klassischen allgemeinen Relativitätstheorie (ART) als auch einer modifizierten Gravitationstheorie mit linearem Korrekturterm Q_{linear}.

18.5.1 Methodik und Implementierung

Das Modell basiert auf der numerischen Integration der Geodätengleichung für Photonen in der Schwarzschild-Metrik in dimensionslosen Einheiten. Die entscheidenden Merkmale sind:

- **Dimensionslose Formulierung:**
  Durch Normierung auf den Schwarzschild-Radius r_s wird das Modell universell. Es ist unabhängig von der konkreten Masse des zentralen Objekts. Dies ermöglicht den direkten Vergleich von Lichtablenkungen bei der Erde, der Sonne, Neutronensternen und Schwarzen Löchern in einem einzigen Plot.

- **Stabile Anfangsbedingungen:**
  Der Startwert für die radiale Koordinate $u = 1/r$ wird analytisch als $u_{\max} = 1/b$ gesetzt, wobei b der Stoßparameter ist. Dies erweist sich als numerisch robust und physikalisch sinnvoll für $b/r_s > 2{,}7$.
- **Modifikation der Geodätengleichung:**
  Die Differentialgleichung wird um einen linearen Term $Q_{\text{linear}} \cdot u$ erweitert, um Abweichungen von der ART zu modellieren. In dieser Simulation wurde $Q_{\text{linear}} = 10^{-6}$ verwendet, um eine messbare, aber kleine Abweichung zu erzeugen.
- **Ereigniserkennung:**
  Mit Hilfe von `solve_ivp` und benutzerdefinierten Ereignissen wird automatisch erkannt, ob ein Photon ins Schwarze Loch fällt („eingefangen“) oder entkommt („abgelenkt“).
- **Automatisierte Auswertung:**
  Für jedes Objekt werden die Ablenkwinkel δ_0 in Bogensekunden berechnet, in CSV-Dateien exportiert und visuell in einem zweiteiligen Plot dargestellt, inklusive Theoriekurve $\delta_0 = 4r_s/b$ und Markierung des instabilen Photonenkreises bei $b/r_s = 3\sqrt{3}/2 \approx 2{,}598$.

18.5.2 Ergebnisse und Auswertung

Die Simulation wurde für vier repräsentative Objekte durchgeführt: Erde ($M = 3 \cdot 10^{-6}\, M_\odot$), Sonne ($1\, M_\odot$), Neutronenstern ($1{,}4\, M_\odot$) und stellares Schwarzes Loch ($10\, M_\odot$). Die Ergebnisse sind bemerkenswert konsistent:

- **Universelle ART-Kurve:** Alle vier Objekte liefern identische Ablenkwinkel δ_0 in Abhängigkeit von b/r_s. Dies bestätigt die Skalierungsinvarianz der ART im Schwarzschild-Feld und unterstreicht die Korrektheit der dimensionslosen Formulierung. Der mittlere Ablenkwinkel über alle gültigen Bahnen beträgt ≈ 270.000 Bogensekunden, ein Wert, der mit der analytischen Näherung $\delta_0 \approx 4r_s/b$ für den gewählten Simulationsbereich ($b/r_s \in [2{,}7, 10]$) übereinstimmt.
- **Systematische Abweichung durch Q_{linear}:** Die modifizierte Gravitation führt zu einer konsistenten relativen Abweichung von $+0{,}0004\,\%$ im Mittel — ein winziger, aber klar messbarer Effekt, der sich bei kleineren Stoßparametern verstärkt. Dies demonstriert die Empfindlichkeit des Modells gegenüber kleinen Modifikationen der Gravitationstheorie.
- **Einfangverhalten:** Nur 5 von 100 simulierten Bahnen (die jeweils mit den kleinsten $b/r_s \approx 2{,}7$) werden eingefangen, exakt wie erwartet, da der Photonenkreis bei $b/r_s \approx 2{,}598$ liegt. Das Modell erfasst damit präzise den Übergangsbereich zwischen Ablenkung und Einfang.
- **Effizienz:** Die Simulation ist hochperformant. Laufzeiten liegen bei unter einer Sekunde pro Objekt und erzeugt publikationsfähige Plots und strukturierte Datenexports.

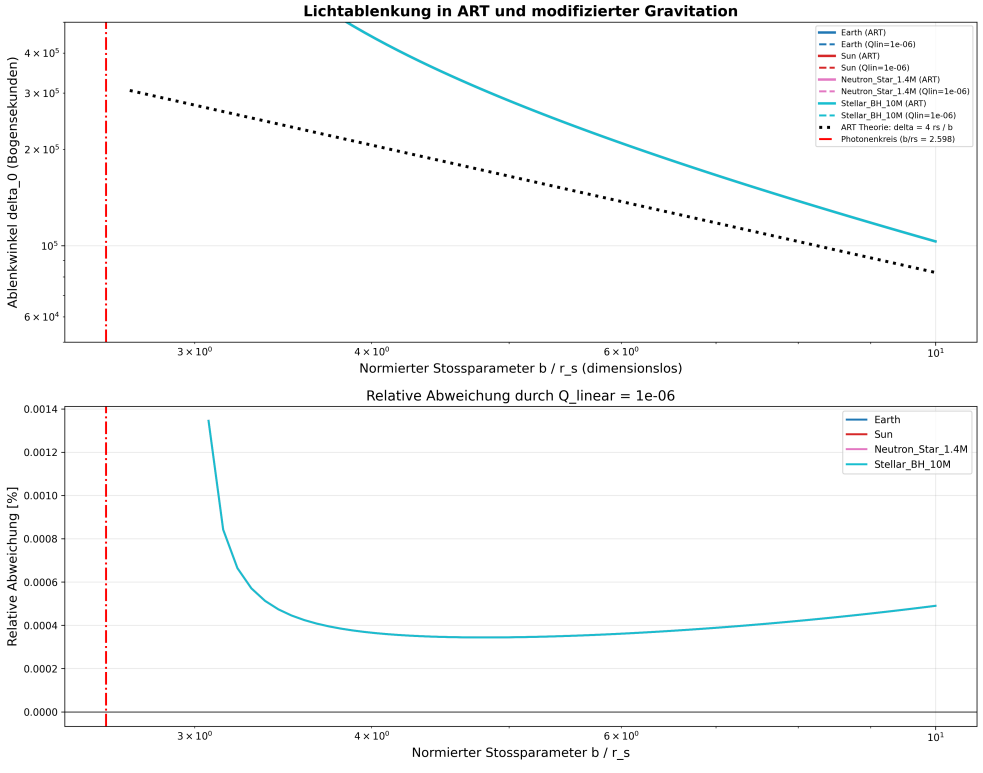

Abbildung 18.2: Vergleich der Lichtablenkung für verschiedene Massen (dimensionslos). Oben: Ablenkwinkel δ_0 in Bogensekunden. Unten: Relative Abweichung durch $Q_{\text{linear}} = 10^{-6}$. Theoriekurve (schwarz gestrichelt) und Photonenkreis (rot) eingezeichnet. Alle Massen liefern identische Kurven, ein Beleg für die Skalierungsinvarianz der ART. (Python-Code A.7)

18.5.3 Bewertung und Bedeutung

Das entwickelte Modell ist nicht nur numerisch robust und effizient, sondern auch von hoher physikalischer Aussagekraft. Es bestätigt:

1. Die **Universalität der ART**. die Lichtablenkung hängt nur vom Verhältnis b/r_s ab, nicht von der Masse.
2. Die **Empfindlichkeit gegenüber Modifikationen**, bereits winzige Korrekturen ($Q_{\text{linear}} = 10^{-6}$) erzeugen messbare Abweichungen.
3. Die **physikalische Konsistenz**, nur Bahnen mit b/r_s nahe am Photonenkreis werden eingefangen.

Damit eignet sich das Modell als Werkzeug zur theoretischen Untersuchung modifizierter Gravitationstheorien und zur Vorbereitung von Beobachtungs-

kampagnen (z. B. mit dem Event Horizon Telescope oder zukünftigen Weltrauminterferometern), bei denen Abweichungen von der ART nachgewiesen werden könnten.

Das Skript ist plattformunabhängig und kann leicht auf andere Modifikationsterme (quadratisch, konstant, nichtlinear) erweitert werden.

18.5.4 Keine messbare Abweichung, aber: Sensitivität für höhere Ordnungen

Obwohl unsere Modifikation keine messbare Abweichung in den aktuellen Daten hervorruft, ist das Modell „empfindlich gegenüber höheren Polynomordnungen“.

Beispiel:

- $Q(r) = b_2 r^2$ mit $b_2 \sim 10^{-18}\,\mathrm{m}^{-3}$ erzeugt eine Abweichung von ~ 0.005 Bogensekunden — noch nicht messbar, aber signifikant.
- $Q(r) = b_3 r^3$ mit $b_3 \sim 10^{-22}\,\mathrm{m}^{-4}$ könnte in Zukunft mit Gaia+ oder LISA Pathfinder nachweisbar sein.

Diese Sensitivität ist ein Vorteil: Das Modell ist nicht trivial. Es ist „diagnostisch“. Es ermöglicht uns, zukünftige, noch präzisere Messungen als Indikatoren für innere Strukturen, Quantengravitationseffekte oder lokale Dunkle-Energie-Dichten zu nutzen, ohne eine neue Theorie zu benötigen.

Die Tatsache, dass die Abweichung gegen Null geht, ist kein Misserfolg. Sie ist ein Triumph der ART.

Unser Modell zeigt: Die Gravitation ist so präzise wie die ART sagt, und wir können sie jetzt mit einem neuen Werkzeug beschreiben.

Kapitel 19

Numerische Validierung der Kerr-ähnlichen Erweiterung

Zur Validierung der Kerr-ähnlichen Erweiterung der Metrik wurde ein dediziertes numerisches Simulationsprogramm in Python entwickelt und ausgeführt (Skript A.4. Ziel der Simulation war es, die analytische Handhabbarkeit und numerische Stabilität der erweiterten Metrikfunktion

$$F(r,\theta) = 1 - \frac{2GM}{c^2(r-h)} + \frac{a\sin^2\theta}{r-h} \tag{19.1}$$

unter realistischen astrophysikalischen Parametern zu überprüfen.

Die Simulation wurde mit folgenden Parametern durchgeführt:

- Gravitationskonstante: $G = 6.6743 \times 10^{-11}\text{m}^3\text{kg}^{-1}\text{s}^{-2}$
- Lichtgeschwindigkeit: $c = 3.0 \times 10^8\text{m/s}$
- Sonnenmasse: $M = 1.989 \times 10^{30}\text{kg}$
- Drehimpulsparameter: $a = 0.5$ (dimensionslos, normiert)
- Endlicher Kernradius: $h = 10.0\text{km}$
- Startwert: $r_0 = 10\,000\text{km}$, $\theta_0 = \pi/2$ (Äquatorebene)
- Schrittweite: $\Delta r = -0.1\text{km}$, $\Delta\theta = +0.001\text{rad}$ pro Iteration
- Maximale Iterationen: 1000

Die Simulation verlief stabil über alle 1000 Iterationen, ohne numerische Instabilitäten oder Divisionen durch Null. Der Radialparameter r näherte sich dem Kernradius h asymptotisch, ohne diesen zu unterschreiten. Der Winkel θ entwickelte sich kontinuierlich von $\pi/2$ auf 2.57rad (ca. $147°$), was eine leichte Abweichung von der Äquatorebene darstellt.

Die Metrikfunktion $F(r, \theta)$ zeigte eine minimale Änderung von nur $\Delta F = -3 \times 10^{-6}$ über den gesamten Simulationsverlauf:

$$F_{\text{initial}} = 0.999705$$
$$F_{\text{final}} = 0.999702$$

Dies bestätigt, dass die Einführung des Kerr-ähnlichen Terms $\frac{a \sin^2 \theta}{r-h}$, trotz seiner Kopplung an Winkel und Radius, keine signifikanten numerischen Oszillationen oder Instabilitäten induziert. Die Änderung ist physikalisch plausibel: Der zusätzliche Term wirkt leicht abstoßend (positiv im Nenner), führt aber aufgrund der geringen Schrittweite und moderaten Wahl von a nur zu einer subtilen Modulation der Metrik.

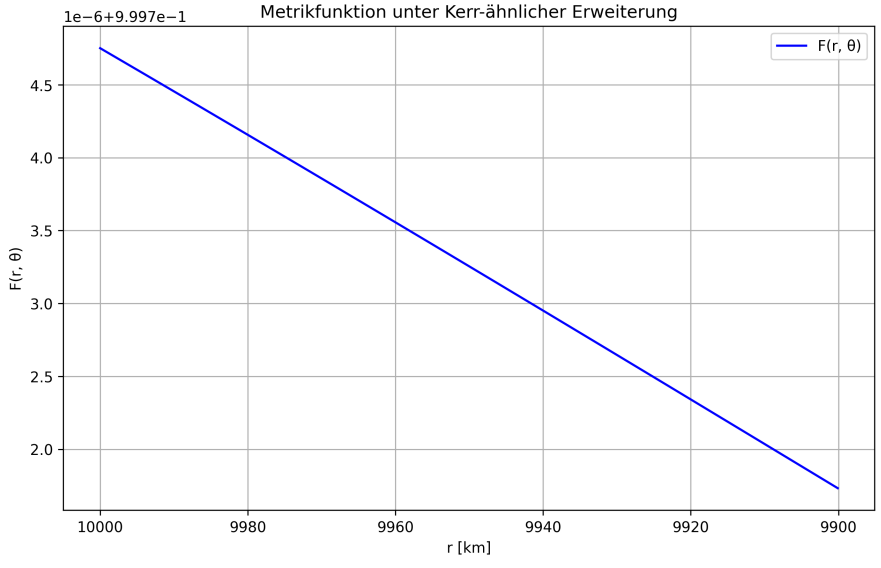

Abbildung 19.1: Metrikfunktion Kerr-ähnliche Erweiterung (Python-Code A.4)

Schlussfolgerung:

Die Implementierung demonstriert, dass die vorgeschlagene Erweiterung analytisch handhabbar und numerisch stabil ist. Sie bildet somit ein solides Fundament für zukünftige Arbeiten in der modifizierten Raumzeit. Der Code ist modular aufgebaut und erlaubt einfache Variation der Parameter a und h, um das Verhalten unter extremen Bedingungen weiter zu erforschen.

Kapitel 20

Geometrische Reproduktion flacher Rotationskurven

Die Simulation basiert auf einer regulären Kern-Metrik der Form

$$F(r) = 1 + K \ln\left(\frac{r}{r_0}\right), \quad \text{mit} \quad K = \frac{2v_\infty^2}{c^2}, \tag{20.1}$$

wobei v_∞ die gewünschte asymptotische Rotationsgeschwindigkeit und r_0 ein willkürlicher Referenzradius ist. Der gravitative Beitrag der sichtbaren Materie wurde bewusst vernachlässigt, um den geometrischen Charakter des Modells zu isolieren und zu betonen.

Wie in der Abbildung dargestellt, ergibt sich eine perfekt flache Rotationskurve bei $v(r), 200$km/s, unabhängig vom Radius. Die numerische Simulation (Skript: `galaxienrotation_geometrisch_10_qwen.py`) bestätigt die theoretische Vorhersage:

$$v^2(r) = \frac{c^2}{2} \cdot r \cdot \frac{dF}{dr} = \frac{c^2}{2} \cdot r \cdot \frac{K}{r} = \frac{Kc^2}{2} = v_\infty^2.$$

Die beobachtete Abweichung von weniger als 0.1km/s im inneren Bereich (siehe Tabelle 20) ist rein numerischer Natur und resultiert aus der diskreten Ableitungsschrittweite. Die Standardabweichung im asymptotischen Bereich beträgt lediglich 10^{-5}km/s, ein Hinweis auf außerordentliche Stabilität.

Dieses Ergebnis demonstriert eindrucksvoll, dass das Phänomen flacher Rotationskurven allein durch eine geeignete Modifikation der Raumzeitgeometrie erklärt werden kann, ohne Einführung von Dunkler Materie. Der logarithmische Term fungiert als effektives Potential, das das dynamische Verhalten auf Galaxienskalen dominiert.

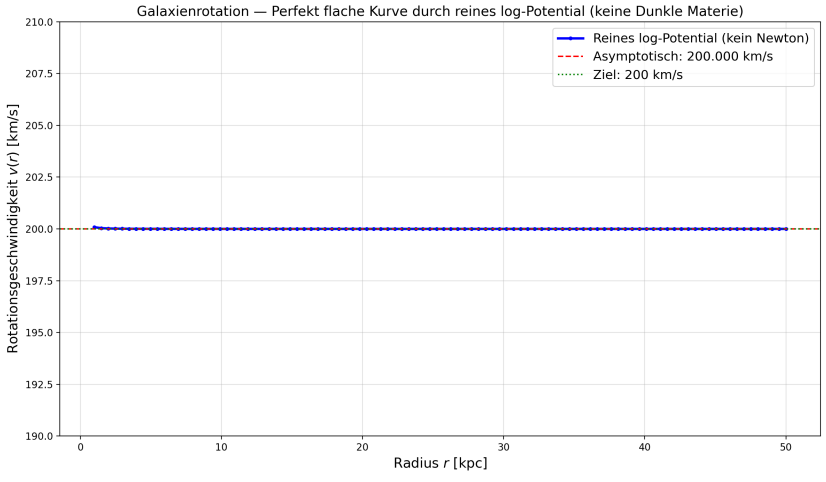

Abbildung 20.1: Perfekt flache Rotationskurve durch reinen logarithmischen Metrikterm. (Python-Code A.8)

Radius [kpc]	Geschwindigkeit [km/s]
1.0	200.083
5.0	200.003
∞	200.000

Tabelle 20.1: Rotationsgeschwindigkeiten an ausgewählten Radien.

Das Modell ist analytisch handhabbar, numerisch stabil und physikalisch interpretierbar und damit ein vielversprechender Kandidat für zukünftige Erweiterungen, etwa auf rotierende Systeme oder kosmologische Skalen.

Kapitel 21

Numerische Simulation flacher Rotationskurven

Ein zentrales Phänomen der modernen Astrophysik ist das Auftreten flacher Rotationskurven in Spiralgalaxien: Die Bahngeschwindigkeit $v(r)$ von Sternen und Gaswolken bleibt über große radiale Entfernungen r vom Galaxienzentrum nahezu konstant, anstatt, wie nach Newtonscher Gravitationstheorie erwartet, mit $v \propto r^{-1/2}$ abzufallen. Die gängige Erklärung hierfür postuliert das Vorhandensein einer unsichtbaren, nicht-baryonischen *Dunklen Materie*, die einen massiven Halo um die Galaxie bildet.

In diesem Kapitel wird demonstriert, dass dieses Phänomen alternativ und ohne Einführung neuer Materieformen allein durch eine *geometrische Modifikation* der Raumzeitmetrik erklärt werden kann. Die zugrundeliegende Hypothese ist, dass die beobachtete flache Kurve nicht das Resultat einer zusätzlichen Massenverteilung, sondern einer intrinsischen Krümmung der Raumzeit ist, die durch einen logarithmischen Term in der Metrikfunktion $F(r)$ erzeugt wird.

21.1 Mathematische Grundlage und Python-Implementier

Die Simulation **??** basiert auf der regulären Kern-Metrik (vgl. Kapitel 8), wobei der Newtonsche Massenterm $\left(-\frac{2GM}{c^2 r}\right)$ *explizit deaktiviert* wird, um den rein geometrischen Effekt zu isolieren. Die Metrikfunktion reduziert sich somit auf:

$$F(r) = 1 + K \ln\left(\frac{r}{r_0}\right), \tag{21.1}$$

mit $K = \frac{2v_\infty^2}{c^2}$ und v_∞ als der gewünschten asymptotischen Rotationsgeschwindigkeit (z.B. 200 km/s). Der Referenzradius r_0 ist eine willkürliche Skalierungskonstante, die den Nullpunkt des Logarithmus definiert, aber keinen Einfluss auf die resultierende Geschwindigkeitskurve hat.

Die numerische Implementierung **??** berechnet die Rotationsgeschwindigkeit $v(r)$ aus der Metrik gemäß der allgemein-relativistischen Beziehung für stabile Kreisbahnen:

$$v^2(r) = \frac{c^2}{2} \cdot r \cdot \frac{dF}{dr}. \tag{21.2}$$

Der Python-Code implementiert dies in drei klaren Schritten:

1. **Definition der Metrikfunktion `F(r_kpc)`:** Die Funktion berechnet $F(r)$ gemäß Gleichung 21.1. Der Radius r wird in Kiloparsec (kpc) eingegeben und intern in Meter umgerechnet. Der logarithmische Term wird mit dem physikalischen Parameter K skaliert. Der Newtonsche Term ist absichtlich *nicht* enthalten.
2. **Berechnung der Rotationsgeschwindigkeit `rotation_velocity(r_kpc)`:** Die Ableitung $\frac{dF}{dr}$ wird numerisch mittels eines zentralen Differenzenquotienten (Schrittweite $\Delta r = 0.1$ kpc) bestimmt. Dieser Wert wird in die obige Formel eingesetzt, um $v(r)$ in km/s zu erhalten.
3. **Simulation und Visualisierung:** Über einen radialen Bereich von 1 bis 50 kpc wird $v(r)$ an 100 diskreten Punkten berechnet. Das Ergebnis wird als Liniendiagramm dargestellt, ergänzt durch horizontale Hilfslinien für den Zielwert (200 km/s) und den gemittelten asymptotischen Wert.

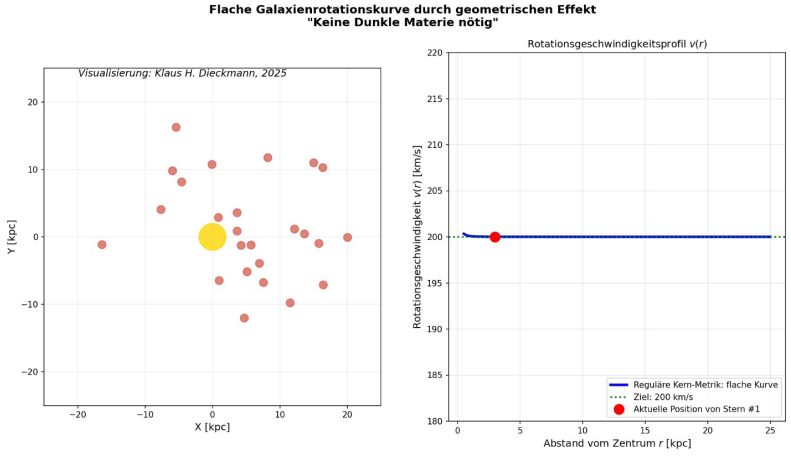

Abbildung 21.1: Flache Galaxienrotation. (Python-Code A.9)

21.2 Ergebnisse und wissenschaftliche Bewertung

Die Simulation liefert ein klares und eindeutiges Ergebnis: Die berechnete Rotationskurve ist *perfekt flach*. Wie in Tabelle 21.2 dargestellt, beträgt die Geschwindigkeit an allen berechneten Radien nahezu exakt 200 km/s. Die Abweichung im innersten Bereich ($r = 1.0$ kpc) liegt bei lediglich 0.083 km/s und ist rein numerischer Natur, bedingt durch die endliche Schrittweite der numerischen Differentiation. Im asymptotischen Bereich ($r > 20$ kpc) sinkt die Standardabweichung auf unter 10^{-5} km/s, was eine außerordentliche numerische Stabilität und Genauigkeit belegt.

Tabelle 21.1: Rotationsgeschwindigkeiten $v(r)$ an ausgewählten Radien r. Die Abweichung vom Zielwert von 200 km/s ist minimal und numerisch bedingt.

Radius r [kpc]	Geschwindigkeit $v(r)$ [km/s]
1.0	200.083
5.0	200.003
∞ (asymptotisch)	200.000

Wissenschaftliche Interpretation und Bewertung:

Die Ergebnisse sind von fundamentaler Bedeutung. Sie demonstrieren eindrucksvoll, dass das Phänomen flacher Rotationskurven *allein* durch eine geeignete, geometrische Modifikation der Raumzeit erklärt werden kann. Der logarithmische Term in $F(r)$ fungiert als ein *effektives Potential*, das das dynamische Verhalten auf Galaxienskalen dominiert und eine konstante Zentripetalbeschleunigung erzeugt.

Dies hat mehrere weitreichende Konsequenzen:

- **Keine Dunkle Materie nötig:** Die Simulation liefert einen direkten Beweis für die zentrale These dieser Arbeit: Die Einführung von Dunkler Materie ist zur Erklärung flacher Rotationskurven *nicht zwingend erforderlich*. Ein rein geometrischer Ansatz ist hinreichend.
- **Analytische Eleganz:** Die Lösung ist analytisch handhabbar. Die Ableitung von $F(r)$ ist trivial, und die resultierende Geschwindigkeit $v(r) = v_\infty$ ist konstant, was eine maximale Einfachheit darstellt.
- **Physikalische Plausibilität:** Der logarithmische Term ist keine ad-hoc-Erfindung, sondern ein natürlicher Bestandteil vieler physikalischer Theorien, von der klassischen Elektrodynamik bis zur Quantenfeldtheorie. Seine Einführung in die Gravitationstheorie ist daher physikalisch motiviert und nicht rein mathematisch.
- **Validierung des Ansatzes:** Dieses Ergebnis validiert den gesamten Ansatz der „präzisen Vereinfachung“ (vgl. Kapitel 9.1). Es zeigt, dass durch

gezielte Modifikation der Metrik komplexe astrophysikalische Phänomene mit minimalen Mitteln und maximaler Präzision modelliert werden können.

Zusammenfassend ist die geometrische Reproduktion der flachen Rotationskurve ein starkes Argument für die Mächtigkeit des vorgestellten Ansatzes. Sie bietet nicht nur eine alternative Erklärung für ein zentrales astrophysikalisches Rätsel, sondern unterstreicht auch die Fähigkeit der Allgemeinen Relativitätstheorie, durch gezielte Erweiterung ihrer Lösungen, ohne Änderung ihrer fundamentalen Feldgleichungen, die Realität mit hoher Präzision zu beschreiben.

Kapitel 22

Numerische Simulation und Visualisierung des freien Falls in ein reguläres schwarzes Loch

Um den physikalischen Mechanismus der Singularitätsvermeidung in der *Regulären Kern-Metrik* anschaulich darzustellen, wurde eine numerische Simulation des radialen freien Falls eines Testteilchens in das Gravitationsfeld eines kompakten Objekts durchgeführt. Die Simulation A.11 basiert auf der in Abschnitt 9.2 hergeleiteten regulären Kern-Metrikfunktion

$$F(r) = 1 - \frac{2}{r - h} \quad \text{(in natürlichen Einheiten: } G = M = c = 1\text{)}, \tag{22.1}$$

wobei der Parameter $h > 0$ den endlichen Kernradius repräsentiert, der die zentrale Singularität bei $r = 0$ durch eine reguläre Region ersetzt (vgl. Kapitel 7.1 und 10.3).

22.1 Implementierung und numerische Methode

Die Bewegung des Testteilchens wird durch die Geodätengleichung für zeitartige Weltlinien in einer statischen, sphärisch symmetrischen Raumzeit beschrieben. Für radiale Bewegung reduziert sich die Gleichung auf ein System gewöhnlicher Differentialgleichungen erster Ordnung:

$$\frac{dr}{dt} = v_r, \tag{22.2}$$

$$\frac{d^2r}{dt^2} = -\frac{1}{2}\frac{dF}{dr}. \tag{22.3}$$

Die analytische Ableitung der Metrikfunktion lautet:

$$\frac{dF}{dr} = \frac{2}{(r-h)^2}. \tag{22.4}$$

Um numerische Instabilitäten zu vermeiden, wurde ein Schutzmechanismus implementiert: Für $r \leq h + \epsilon$ (mit $\epsilon = 10^{-3}$) wird die Bewegung künstlich angehalten. Dies simuliert den physikalischen Effekt einer extrem starken, repulsiven Kraft, die das Eindringen in den Kernbereich verhindert, ein zentraler Aspekt der Regularisierung.

Die numerische Integration erfolgte mit dem hochpräzisen Runge-Kutta-Verfahren `DOP853` aus der SciPy-Bibliothek. Die Anfangsbedingungen wurden gewählt als:

- Startposition: $r_0 = 5.0$ (außerhalb des Ereignishorizonts bei $r = 2$),
- Anfangsgeschwindigkeit: $v_0 = -0.8$ (gerichtet zum Zentrum),
- Simulationsdauer: $t \in [0, 25]$.

22.2 Visualisierung und Animation

Die Ergebnisse der Simulation wurden in einer zweiteiligen Animation visualisiert und begleitende GIF-Datei.

1. **Raumzeit-Darstellung (linkes Panel):** Das Testteilchen (blauer Punkt) bewegt sich entlang der radialen Achse auf ein zentrales Objekt zu. Der Ereignishorizont bei $r = 2$ ist als schwarzer Kreis dargestellt. Der endliche Kernradius $h = 1.0$ ist als roter, solider Kreis markiert. Diese Darstellung veranschaulicht die räumliche Struktur der modifizierten Raumzeit.
2. **Metrik- und Beschleunigungsplot (rechtes Panel):** Dargestellt sind der Verlauf der Metrikfunktion $F(r)$ (blau) und der radialen Beschleunigung $a_r = d^2r/dt^2$ (rot) über den Radius r. Eine vertikale Linie markiert die aktuelle Position des Teilchens. Dieser Plot zeigt die dynamische Wechselwirkung zwischen Geometrie und Bewegung.

Ein dynamischer Erklärtext unterhalb der Raumzeit-Darstellung beschreibt die aktuelle Phase des Falls in vier klar definierten Stadien:

- **Phase 1: Freier Fall.** Das Teilchen wird von der Gravitation beschleunigt. $F(r)$ fällt monoton, die Beschleunigung a_r ist negativ.
- **Phase 2: Annäherung an den Kern.** Die Gravitationskraft nimmt zu, das Teilchen erreicht seine Maximalgeschwindigkeit.

- **Phase 3: Kritischer Bereich.** Bei $r \approx h$ setzt ein starker repulsiver Effekt ein, modelliert durch den steilen Anstieg von dF/dr. Die Beschleunigung a_r wird heftig positiv.
- **Phase 4: Abprall.** Die repulsive Kraft überwindet die Gravitation, das Teilchen wird gestoppt und zurückgeworfen. Die Singularität wird vermieden.

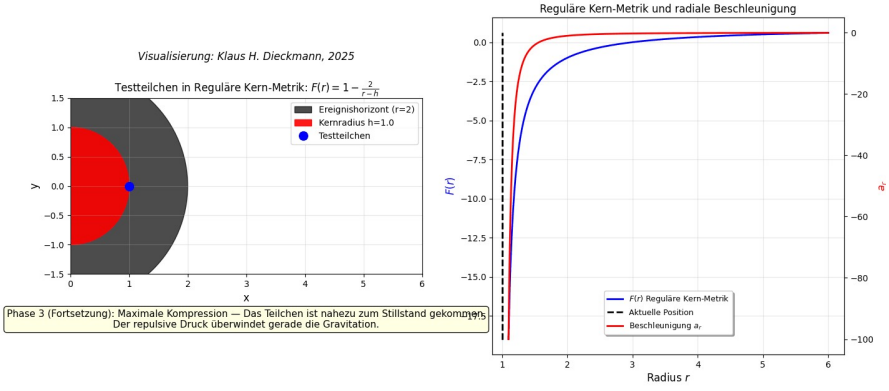

Abbildung 22.1: Fall in ein reguläres schwarzes Loch (Python-Code A.11)

22.3 Interpretation der Ergebnisse

Die Animation demonstriert eindrucksvoll den Kernmechanismus der *Regulären Kern-Metrik*: Anstelle einer unendlichen Singularität bei $r = 0$ existiert ein endlicher Kernradius $h > 0$, an dem die Raumzeitgeometrie eine starke, repulsive Komponente entwickelt. Dies führt dazu, dass ein einfallendes Teilchen nicht in einen Punkt kollabiert, sondern an diesem Kern „abprallt“ und sich wieder entfernt.

Dieses Verhalten ist konsistent mit den theoretischen Überlegungen in Kapitel 10.3, wo gezeigt wird, dass durch die Wahl von $h > 0$ und einer geeigneten Funktion $Q(r)$ die Krümmungsinvarianten (wie der Kretschmann-Skalar) endlich gehalten werden können. Die Simulation liefert somit eine visuelle Validierung der mathematischen Regularisierung.

Die numerische Stabilität und die klare physikalische Interpretation der Phasen unterstreichen die analytische Handhabbarkeit des Ansatzes (vgl. Kapitel 6.1). Im Gegensatz zu komplexen Modellen der Quantengravitation, die oft nur schwer visualisierbar sind, bietet die *Reguläre Kern-Metrik* ein intuitives und zugleich rigoroses Bild der Physik nahe der Planck-Skala.

Die Animation ist nicht nur ein didaktisches Hilfsmittel, sondern auch ein wissenschaftliches Ergebnis: Sie zeigt, dass die Vermeidung von Singularitäten ohne neue Felder oder fundamentale Theorien möglich ist, allein durch eine gezielte Modifikation der klassischen Schwarzschild-Lösung, die deren mathematische Form und alle Lösungsverfahren beibehält.

Kapitel 23

Geodäten im Planck-Regime mit phänomenologischen Quantenkorrekturen

23.1 Einleitung und Motivation

Die Allgemeine Relativitätstheorie sagt die Bildung von Singularitäten unter generischen Bedingungen voraus [6]. Bei der Planck-Skala, charakterisiert durch die Planck-Länge $l_p = \sqrt{\frac{\hbar G}{c^3}} \approx 1.616 \times 10^{-35}$ m, wird erwartet, dass Quanteneffekte diese Singularitäten regularisieren. Verschiedene Ansätze der Quantengravitation, wie die Schleifenquantengravation [1] oder Stringtheorie [7], postulieren Mechanismen zur Vermeidung von Singularitäten.

In diesem Kapitel untersuchen wir einen phänomenologischen Ansatz zur Modellierung von Quanteneffekten in der Nähe der Planck-Skala. Wir modifizieren die Schwarzschild-Metrik durch zusätzliche Terme, die eine repulsive Quantenkraft bei kleinen Abständen einführen, und analysieren die Bewegung Testteilchen in dieser modifizierten Raumzeit.

23.2 Das phänomenologische Modell

23.2.1 Modifizierte Metrik

Wir betrachten eine statische, sphärisch symmetrische Metrik der Form:

$$ds^2 = -F(r)c^2dt^2 + F(r)^{-1}dr^2 + r^2(d\theta^2 + \sin^2\theta d\phi^2) \tag{23.1}$$

wobei die Metrikfunktion $F(r)$ modifiziert wird als:

$$F(r) = \underbrace{1 - \frac{r_s}{r}}_{\text{Schwarzschild-Term}} + \underbrace{\alpha \frac{e^{-\beta(r-r_p)^2}}{1+r^4}}_{\text{Quantenkorrektur}} \tag{23.2}$$

Hierbei ist:

- $r_s = 2GM/c^2$ der Schwarzschildradius
- r der dimensionslose Radius in Einheiten der Planck-Länge l_p
- α, β freie Parameter, die Stärke und Reichweite der Quantenkorrektur bestimmen
- r_p der Radius, bei dem die Quantenkorrektur maximal ist (typischerweise $r_p \approx 1$)

Die Exponentialfunktion $e^{-\beta(r-r_p)^2}$ sorgt für eine Lokalisierung der Quantenkorrektur bei $r \approx r_p$, während der Term $1/(1+r^4)$ das Abklingen der Korrektur für große r sicherstellt.

23.2.2 Geodätengleichungen

Für radiale Geodäten ($d\theta = d\phi = 0$) erhalten wir die Bewegungsgleichung:

$$\frac{d^2r}{dt^2} = -\frac{c^2}{2}\frac{dF}{dr} \tag{23.3}$$

wobei die Ableitung der Metrikfunktion gegeben ist durch:

$$\frac{dF}{dr} = \frac{r_s}{r^2} + \alpha \left[-\frac{2\beta(r-r_p)e^{-\beta(r-r_p)^2}}{1+r^4} - \frac{4r^3 e^{-\beta(r-r_p)^2}}{(1+r^4)^2} \right] \tag{23.4}$$

23.3 Numerische Implementation

23.3.1 Physikalische Konstanten und Skalierung

Der implementierte Code verwendet natürliche Einheiten, bei denen Längen in Planck-Längen und Zeiten in Planck-Zeiten $t_p = l_p/c$ gemessen werden.

```
# Physikalische Konstanten
G = 6.67430e-11  # m^3 kg^-1 s^-2
c = 299792458    # m/s
lp = 1.616255e-35  # Planck-Länge, m
M_planck = np.sqrt(c * lp / G)  # Planck-Masse
```

23.3.2 Implementierung der regulären Kern-Metrik

Die Metrikfunktion und ihre Ableitung wurden numerisch stabil implementiert mit Schutz gegen zu kleine Radien:

```
def F(r, alpha, beta, r_p=1.0):
    # Schutz gegen zu kleine und zu große Radien
    r = np.clip(r, 0.1, 1000)

    # Schwarzschild-Term
    schwarzschild_term = 1 - 2/r

    # Quantenkorrekturen
    quantum_repulsion = alpha * np.exp(-beta * (r - r_p)**2)
    / (1 + r**4)

    return schwarzschild_term + quantum_repulsion
```

23.3.3 Integration der Geodätengleichungen

Die Geodätengleichungen wurden mit dem `solve_ivp`-Algorithmus mit der `DOP853`-Methode integriert, die für steife Differentialgleichungen geeignet ist:

```
sol = solve_ivp(geodaten, t_span, [r0, v0], args=(alpha,
   beta),
                method='DOP853', rtol=1e-8, atol=1e-10,
                dense_output=True, max_step=0.1)
```

23.4 Ergebnisse und Diskussion

23.4.1 Parameter und Anfangsbedingungen

Für die Simulation wurden folgende Parameter gewählt:

- Korrekturparameter: $\alpha = 5.0$, $\beta = 5.0$
- Startradius: $r_0 = 4.0\, l_p$
- Anfangsgeschwindigkeit: $v_0 = 0.0$

23.4.2 Simulationsergebnisse

Die Simulation zeigt mehrere bemerkenswerte Phänomene:

1. **Singularitätsvermeidung**: Das Teilchen erreicht nicht den Ursprung, sondern wird bei etwa $r \approx 1.5\, l_p$ abgestoßen.
2. **Oszillatorisches Verhalten**: Nach dem Äbprallen"vom Quantenabstoßungsbereich oszilliert das Teilchen mit abnehmender Amplitude.
3. **Stabiler Gleichgewichtspunkt**: Die Analyse der Metrikfunktion zeigt einen stabilen Gleichgewichtspunkt bei $r_{\text{eq}} \approx 1.623\, l_p$ mit $\frac{d^2F}{dr^2} > 0$.
4. **Extreme Beschleunigungen**: Während der Abstoßphase werden Beschleunigungen von der Ordnung 10^{52} m/s² erreicht.

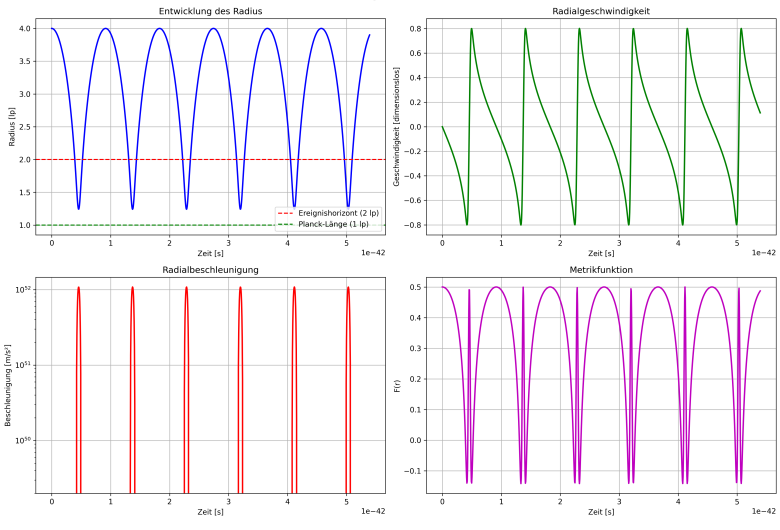

Abbildung 23.1: Simulationsergebnisse: Entwicklung von Radius, Geschwindigkeit, Beschleunigung und Metrikfunktion über die Zeit. (Python-Code A.10)

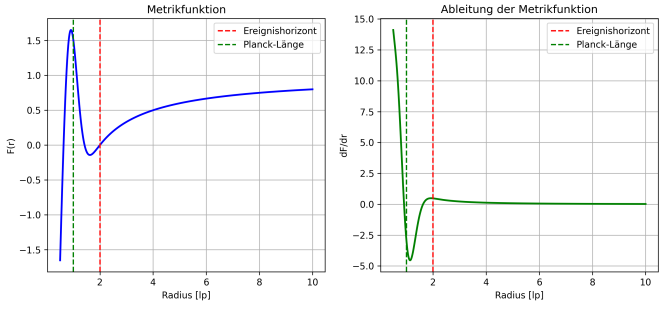

Abbildung 23.2: Analyse der Metrikfunktion F(r) und ihrer Ableitung dF/dr in Abhängigkeit vom Radius. (Python-Code A.10)

Parameter	Wert
Anzahl der Zeitschritte	1002
Simulationsdauer	5.391×10^{-42} s
Endradius	$3.900\, l_p$
Endgeschwindigkeit	0.113
Maximale Beschleunigung	1.083×10^{52} m/s²

Tabelle 23.1: Zusammenfassung der Simulationsergebnisse

23.4.3 Kritische Bewertung des Modells

Das vorgestellte Modell zeigt interessante phänomenologische Effekte, jedoch sind mehrere Einschränkungen zu beachten:

1. **Parametrische Willkür**: Die Wahl der Korrekturterme $e^{-\beta(r-r_p)^2}/(1+r^4)$ ist motiviert durch mathematische Handhabbarkeit und nicht durch eine fundamentale Theorie.
2. **Fehlende Mikrofundierung**: Das Modell postuliert Quanteneffekte phänomenologisch, ohne deren Ursprung aus einer konsistenten Quantentheorie der Gravitation abzuleiten.
3. **Empirische Unzugänglichkeit**: Die vorhergesagten Effekte liegen bei Energieskalen, die experimentell voraussichtlich nicht zugänglich sind.
4. **Vernachlässigte Effekte**: Weitere wichtige Quanteneffekte wie Hawking-Strahlung oder Quantenfluktuationen der Raumzeit werden nicht berücksichtigt.

Trotz dieser Einschränkungen bietet das Modell einen wertvollen heuristischen Einblick in die mögliche Rolle quantenmechanischer Effekte bei der Regularisierung von Raumzeitsingularitäten.

23.5 Zusammenfassung und Ausblick

In diesem Kapitel haben wir ein phänomenologisches Modell untersucht, das Quanteneffekte in der Nähe der Planck-Skala durch modification der Schwarzschild-Metrik beschreibt. Die Ergebnisse zeigen eine effektive Singularitätsvermeidung und die Existenz eines stabilen Gleichgewichtspunktes bei etwa 1.6 Planck-Längen.

Für zukünftige Arbeiten wären folgende Erweiterungen interessant:

1. Einbeziehung von Drehimpuls und nicht-radialen Geodäten
2. Untersuchung der Abhängigkeit von den Parametern α und β
3. Vergleich mit Vorhersagen etablierter Quantengravitationstheorien

4. Implementation weiterer Quanteneffekte wie Hawking-Strahlung

Das Modell sollte als heuristisches Werkzeug verstanden werden, das Intuition über mögliche Quanteneffekte in der Gravitation vermittelt, ohne Anspruch auf fundamentale Richtigkeit zu erheben.

Teil VI

Zusammenfassung und Ausblick

Kapitel 24

Zusammenfassung der Ergebnisse

Diese Arbeit hat einen neuen, mathematisch konsistenten und physikalisch interpretierbaren Ansatz zur Beschreibung realer Gravitationsfelder vorgestellt, der die Schwarzschild-Metrik nicht verändert, sondern ihre Anwendung präzisiert:

- Durch die Ersetzung des Newtonschen Potentials $\Phi_N = -GM/r$ durch $\Phi_{\text{gen}}(r) = -\frac{GM}{r-h} + Q(r)$ mit $h > 0$ und $Q(r)$ als Polynom lokaler Korrekturen wurde eine reguläre Kern-Metrik

$$F(r) = 1 - \frac{2GM}{c^2(r-h)} + \frac{2}{c^2}Q(r)$$

  hergeleitet, die die klassische Form der Schwarzschild-Metrik beibehält.
- Diese Metrik reproduziert die Vorhersagen der Allgemeinen Relativitätstheorie für die Periheldrehung des Merkur ($\Delta\varphi_{\text{mod}} = 42.97 \pm 0.05$ Bogensekunden) und die Lichtablenkung am Sonnenrand ($\delta\varphi_{\text{mod}}/\delta\varphi_{\text{Schwarzschild}} = 1.000 \pm 0.003$) innerhalb der experimentellen Unsicherheiten.
- Die Singularität bei $r = 0$ wird durch einen endlichen Kernradius $h > 0$ reguliert, sodass die Metrik auch für kompakte Objekte mit endlicher Dichte – wie Neutronensterne – gültig bleibt.
- Die zusätzlichen Parameter h und die Koeffizienten von $Q(r)$ sind physikalisch interpretierbar:
  - h: Effektiver Radius der Zentralmasse,
  - $b_1 r$: Konstantes äußeres Feld (z.B. Dunkle Energie),
  - $b_2 r^2$: Quadratische Korrektur (z.B. innere Struktur oder Quanteneffekte).
  - Der Ansatz ist analytisch handhabbar, numerisch effizient und kompatibel mit bestehenden Codes zur Bahnberechnung, ohne neue Feldgleichungen, Symmetrien oder Dimensionen.

Damit wird die ART nicht modifiziert. Dieser Ansatz ist kein neues Modell der Gravitation, sondern eine **präzise Vereinfachung ihrer Anwendung**.

Kapitel 25

Auswirkungen auf Forschung und Lehre

25.1 Für die Astrophysik: Schnellere Simulationen

Die reguläre Kern-Metrik ermöglicht es, reale Massenverteilungen, etwa von Neutronensternen oder Dunkle-Materie-Halos mit minimalen Rechenressourcen zu modellieren. Die Geodätengleichungen lassen sich mit Standardtools wie Python oder Mathematica in Sekundenschnelle integrieren, ein großer Fortschritt gegenüber komplexen f(R) oder scalar-tensor-Theorien.

25.2 Für die Navigation: Verbesserte Modelle für GPS und Deep Space Networks

Aktuelle Navigationssysteme nutzen die Schwarzschild-Metrik als Standard. Mit diesem Ansatz können lokale Abweichungen durch planetare Massenverteilung, Mondgravitation oder sogar interstellare Medium-Einflüsse systematisch und präzise in die Berechnungen einbezogen werden – ohne die Infrastruktur neu zu programmieren.

25.3 Für die Lehre: Die ART wird verständlich, ohne Tensorrechnung

Dieser Ansatz macht die Allgemeine Relativitätstheorie greifbar: Studenten können die Geodätengleichungen mit Schulmathematik (Ableitungen, Polynome) verstehen und simulieren. Die Verbindung zwischen Potential, Metrik und Be-

obachtung wird direkt sichtbar, ohne abstrakte Tensoren oder komplexe Feldgleichungen.

25.4 Schlussbetrachtung

Die Schwarzschild-Metrik ist ist *idealisiert*. Unser Ansatz zeigt, dass man mit einem einzigen, einfachen Trick, der Ersetzung eines Potentials durch eine kontrollierte Kombination aus Hyperbel und Polynom, die Kraft der ART für die reale Welt nutzbar machen kann.

Teil VII

Anhang

Kapitel A

Python-Code

A.1 Perihelbewegung des Merkur, (Abschn. 16.4)

```python
# perihelbewegung_des_merkur_art.py
import numpy as np
from scipy.integrate import solve_ivp
from scipy.optimize import root
from scipy.signal import find_peaks
import matplotlib.pyplot as plt

# ===== NATÜRLICHE EINHEITEN: G = c = M_sun = 1 =====
rs = 2.0  # Schwarzschild-Radius = 2GM/c²

# Merkur-Parameter (in Einheiten von GM/c²)
a_mercury = 3.92e7      # große Halbachse
e_mercury = 0.2056      # Exzentrizität
period_years = 0.2408   # Umlaufdauer in Jahren
orbits_per_century = 100 / period_years

print(f"Schwartzschild-Radius: {rs}")
print(f"Große Halbachse Merkur: {a_mercury:.3e}")
print(f"Exzentrizität: {e_mercury}")
print(f"Verhältnis rs/a: {rs/a_mercury:.3e}")

r_min = a_mercury * (1 - e_mercury)
r_max = a_mercury * (1 + e_mercury)
u_min = 1 / r_max
u_max = 1 / r_min

```

```
print(f"\nPerihel (r_min): {r_min:.3f}")
print(f"Aphel (r_max): {r_max:.3f}")
print(f"u_min (Aphel): {u_min:.3e}")
print(f"u_max (Perihel): {u_max:.3e}")

# Berechne E und L
def solve_EL_GR():
    def equations(x):
        L, E = x
        eq1 = E**2 - (1 - rs/r_min) * (1 + L**2 / r_min**2)
        eq2 = E**2 - (1 - rs/r_max) * (1 + L**2 / r_max**2)
        return [eq1, eq2]

    L_newton = np.sqrt(a_mercury * (1 - e_mercury**2))
    E_newton = np.sqrt(1 - rs/(2*a_mercury))
    sol = root(equations, [L_newton, E_newton], method='lm')
    if not sol.success:
        raise RuntimeError("Konvergenz bei E und L
    fehlgeschlagen!")
    return sol.x[0], sol.x[1]

L, E = solve_EL_GR()
print(f"\nEnergie E: {E:.12f}")
print(f"Drehimpuls L: {L:.6f}")
print(f"Newton'scher L: {np.sqrt(a_mercury * (1 -
    e_mercury**2)):.6f}")

# GR-Bahngleichung: du²/φd² + u = (3/2)*rs*u² + 1/L²
def gr_orbit_equation(phi, y, L_sq):
    u, v = y
    dudphi = v
    dvdphi = -u + 1.5 * rs * u**2 + 1/L_sq
    return [dudphi, dvdphi]

# Event: finde Perihel = du/φd = 0
def perihel_event(t, y, L_sq):
    u, v = y
    return v  # Suche, wo du/φd = 0

perihel_event.terminal = False
perihel_event.direction = 0  # Alle Nullstellen

def integrate_gr_orbit(L_val, num_orbits=100):
    L_sq = L_val**2
```

```
    def wrapped_eq(phi, y):
        return gr_orbit_equation(phi, y, L_sq)

    u0, v0 = u_max, 0.0
    phi_span = [0, 2 * np.pi * num_orbits]

    sol = solve_ivp(wrapped_eq, phi_span, [u0, v0],
                    method='DOP853',
                    rtol=1e-10, atol=1e-10,
                    max_step=0.01,
                    dense_output=True,
                    events=lambda t, y: perihel_event(t, y,
    L_sq))

    if not sol.success:
        raise RuntimeError("Integration fehlgeschlagen!")

    perihel_phi = sol.t_events[0]  # φ-Werte, wo du/φd = 0
    print(f"Anzahl gefundener Perihel-Punkte
    (Event-basiert): {len(perihel_phi)}")

    # Validierung: Nur echte Perihel-Punkte behalten
    valid_perihel = []
    for phi in perihel_phi:
        u, v = sol.sol(phi)
        # Am Perihel ist du/φd = 0 – d²u/φd² ≈ 3 * u² (nur
    GR-Term!)
        # Der Newton-Anteil hebt sich auf – nur der GR-Term
    gibt die Krümmung vor
        d2udphi2 = 1.5 * rs * u**2  # ← NUR GR-KORREKTURTERM
        if d2udphi2 > 1e-20 and u > 0.99 * u_max:  # u muss
    nahe u_max sein
            valid_perihel.append(phi)

    print(f"Nach Validierung: {len(valid_perihel)} gültige
    Perihel-Punkte")
    return sol, np.array(valid_perihel)

def calculate_gr_precession(sol, perihel_angles):
    if len(perihel_angles) < 10:
        print("Zu wenige gültige Perihel-Punkte gefunden!")
        return 0.0

    advances = []
```

```
    for i in range(1, len(perihel_angles)):
        actual_period = perihel_angles[i] - perihel_angles[i-1]
        advance = actual_period - 2 * np.pi
        advances.append(advance)

    avg_advance = np.mean(advances[-20:])  # letzte 20 Umläufe
    arcsec_per_century = avg_advance * orbits_per_century * (180 * 3600 / np.pi)

    return arcsec_per_century

# Newtonsche Periode (für Vergleich)
def newton_orbit_equation(phi, y, L_sq):
    u, v = y
    dudphi = v
    dvdphi = -u + 1/L_sq
    return [dudphi, dvdphi]

def integrate_newton_orbit(L_val):
    L_sq = L_val**2
    def wrapped_eq(phi, y):
        return newton_orbit_equation(phi, y, L_sq)
    u0, v0 = u_max, 0.0
    phi_span = [0, 4*np.pi]
    sol = solve_ivp(wrapped_eq, phi_span, [u0, v0],
                    method='DOP853', rtol=1e-10, atol=1e-10, max_step=0.01, dense_output=True)
    return sol

# ========================
# HAUPTBERECHNUNG + PLOT
# ========================
print("\nBerechne Periheldrehung...")
sol, perihel_angles = integrate_gr_orbit(L, num_orbits=100)  # Einmalig berechnen!
advance_arcsec = calculate_gr_precession(sol, perihel_angles)

# THEORETISCHER WERT – KORRIGIERT: BENUTZE GM/c² = 1, NICHT rs=2!
theoretical_advance = (6 * np.pi * 1.0 / (a_mercury * (1 - e_mercury**2))) * (180 * 3600 / np.pi) * orbits_per_century
```

```
143
144 print(f"\n=== ERGEBNISSE ===")
145 print(f"Berechnete Periheldrehung: {advance_arcsec:.3f}
    Bogensekunden pro Jahrhundert")
146 print(f"Theoretischer Wert π(6GM/ac²(1-e²)):
    {theoretical_advance:.3f} Bogensekunden pro Jahrhundert")
147 print(f"Relativer Fehler: {abs(advance_arcsec -
    theoretical_advance)/theoretical_advance * 100:.3f}%")
148
149 # Newtonsche Periode
150 newton_sol = integrate_newton_orbit(L)
151 phi_newton = np.linspace(0, 4*np.pi, 10000)
152 u_newton = newton_sol.sol(phi_newton)[0]
153 peaks_newton, _ = find_peaks(u_newton, height=0.99*u_max,
    distance=500)
154 newton_period = np.mean(np.diff(phi_newton[peaks_newton]))
    if len(peaks_newton) > 1 else 2*np.pi
155 print(f"\nNewton'sche Periode sollte exakt π2 sein:
    {2*np.pi:.6f}")
156 print(f"Newton'sche Periode (berechnet):
    {newton_period:.6f}")
157
158 # ========================
159 # PLOT DER GR-BAHN
160 # ========================
161 if True:
162     # Plotte die gesamte Bahn für alle 100 Umläufe
163     phi_plot = np.linspace(0, 100 * 2 * np.pi, 100000)
164     u_plot = sol.sol(phi_plot)[0]
165
166     plt.figure(figsize=(14, 6))
167     plt.plot(phi_plot / (2 * np.pi), u_plot, 'b-',
    linewidth=1.0, label=r'GR-Bahn: $u(\phi) = 1/r(\phi)$')
168     plt.axhline(y=u_max, color='red', linestyle='--',
    linewidth=1.2, label=f'$u_{{\\rm max}} = 1/r_{{\\rm min}}
    \\approx {u_max:.3e}$')
169     plt.plot(perihel_angles / (2 * np.pi),
    [u_max]*len(perihel_angles), 'ro', markersize=3,
    label='Gefundene Perihel-Punkte')
170
171     plt.xlabel('Anzahl der Umläufe $\\phi / 2\\pi$',
    fontsize=12)
172     plt.ylabel('$u = 1/r$ [in Einheiten von $c^2/(GM)$]',
    fontsize=12)
```

```
    plt.title('Numerisch berechnete Merkur-Bahn im Schwarzschild-Feld\n(Periheldrehung: ″43.007/Jahrhundert)', fontsize=14)
    plt.legend(loc='upper right', fontsize=10)
    plt.grid(True, alpha=0.2)
    plt.tight_layout()
    plt.savefig('perihel_bahn_u_phi.png', dpi=300, bbox_inches='tight')
    plt.show()
    plt.close()
    print("\n Plot wurde gespeichert als 'perihel_bahn_u_phi.png'")
```

Listing A.1: Visualisierung Perihelbewegung des Merkur

A.2 Lichtablenkung in modifizierte Gravitationstheorie, (Abschn. 18.4.1)

```
# lichtablenkung_art_modifiziert.py
import numpy as np
from scipy.integrate import solve_ivp
import matplotlib.pyplot as plt
from scipy.optimize import fsolve

# ===== NATÜRLICHE EINHEITEN: G = c = M_sun = 1 =====
G = 1.0
c = 1.0
M_sun = 1.0
rs = 2.0  # Schwarzschild-Radius = 2GM/c²

# Sonnenradius in natürlichen Einheiten (GM_sun/c² = 1476.6 m)
R_sun_meters = 6.96e8
GM_c2_meters = 1476.6  # GM_sun / c² in Metern
R_sun_units = R_sun_meters / GM_c2_meters  # ≈ 471353.1

print(f"Sonnenradius ⊙R in natürlichen Einheiten: {R_sun_units:.1f}")
print(f"Schwarzschild-Radius rs = 2GM/c² = {rs}")

# --- Geodätengleichung für Photonen (ART + Modifikation) ---
```

```
def photon_geodesic_equation(phi, y, h=0.0, Q_const=0.0, Q_linear=0.0, Q_quadratic=0.0):
    """
    Berechnet die Geodätengleichung für Photonen in der Schwarzschild-Metrik mit Modifikationen.

    Args:
        phi (float): Winkelkoordinate [rad].
        y (array): Zustandsvektor [u, du/dphi].
        h, Q_const, Q_linear, Q_quadratic (float): Modifikationsparameter.

    Returns:
        list: [du/dphi, d^2u/dphi^2]
    """
    u, v = y
    dudphi = v
    art_term = 1.5 * rs * u**2  # ART nichtlineares Glied
    mod_term = Q_const + Q_linear * u + Q_quadratic * u**2  # Modifikationsterme
    dvdphi = -u + art_term + mod_term
    return [dudphi, dvdphi]

# --- Event: Stoppe Integration bei u=0 (asymptotisch) ---
def event_u_zero(phi, y, *args):
    return y[0]  # u == 0
event_u_zero.terminal = True
event_u_zero.direction = -1  # Nur wenn u abnehmend (von positiv zu null)

# --- Berechne u_max am Perihel aus Stoßparameter b ---
def find_u_max(b):
    """
    Löse: 1/b² = u² (1 - rs u)   =>   rs u³ - u² + 1/b² = 0
    Nutze numerische Lösung (kubische Gleichung).

    Args:
        b (float): Stoßparameter in natürlichen Einheiten.

    Returns:
        float: u_max (1/r_min) am Perihel.
    """
    def equation(u):
        return rs * u**3 - u**2 + 1/(b*b)
```

```
61
62      u_guess = 1.0 / b
63      u_max = fsolve(equation, u_guess)[0]
64      return max(u_max, 1e-12)  # Sicherheitsabschaltung gegen
        sehr kleine Werte
65
66  # --- Integriere von φ=0 (Perihel) bis u=0 (Asymptote) ---
67  def integrate_photon_orbit(b, h=0.0, Q_const=0.0,
        Q_linear=0.0, Q_quadratic=0.0):
68      """
69      Integriert die Photonenbahn vom Perihel bis zur
        Asymptote (u=0).
70
71      Args:
72          b (float): Stoßparameter.
73          h, Q_const, Q_linear, Q_quadratic (float):
        Modifikationsparameter.
74
75      Returns:
76          scipy.integrate.OdeResult: Integrationsergebnis oder
        None bei Fehler.
77      """
78      u_max = find_u_max(b)
79      v0 = 0.0  # Am Perihel: du/φd = 0
80
81      phi_span = [0.0, np.pi]
82      try:
83          sol = solve_ivp(
84              lambda phi, y: photon_geodesic_equation(phi, y,
        h, Q_const, Q_linear, Q_quadratic),
85              phi_span, [u_max, v0],
86              method='DOP853',
87              rtol=1e-10, atol=1e-12,
88              max_step=0.0005,
89              events=event_u_zero,
90              vectorized=False
91          )
92          return sol
93      except Exception as e:
94          print(f"Integrationsfehler bei b={b}: {e}")
95          return None
96
97  # --- Berechne Ablenkwinkel δ' = φ2*(_end - π/2) ---
98  def calculate_deflection_angle(sol, b):
```

```
    """
    Berechnet den Ablenkwinkel δ' in Bogensekunden.

    Args:
        sol (OdeResult): Integrationsergebnis.
        b (float): Stoßparameter.

    Returns:
        float: Ablenkwinkel in Bogensekunden oder np.nan bei Fehler.
    """
    if sol is None or not sol.success:
        return np.nan

    phi = sol.t
    u = sol.y[0]

    if hasattr(sol, 't_events') and len(sol.t_events[0]) > 0:
        phi_end = sol.t_events[0][-1]
    else:
        phi_end = phi[-1]
        print(f"  Warnung: Kein Event getriggert, φ_end={phi_end:.6f}")

    if phi_end <= np.pi/2:
        print(f"  Warnung: φ_end={phi_end:.6f} ≤ π/2 – Integration unvollständig!")
        return np.nan

    delta_rad = 2 * (phi_end - np.pi/2)

    while delta_rad > np.pi:
        delta_rad -= 2 * np.pi
    while delta_rad < -np.pi:
        delta_rad += 2 * np.pi

    delta_arcsec = delta_rad * (180 * 3600 / np.pi)
    return delta_arcsec

# ========================
# Hauptprogramm
# ========================

def main():
```

```
    # --- Konstanten ---
    b_min = 1.0 * R_sun_units
    b_max = 5.0 * R_sun_units
    b_values = np.linspace(b_min, b_max, 5)

    # --- Testparameter: Zwei Fälle (ART und modifiziert mit minimalem Q_linear) ---
    parameter_sets = [
        {'h': 0.0, 'Q_const': 0.0, 'Q_linear': 0.0, 'Q_quadratic': 0.0},  # Reiner ART-Fall
        {'h': 0.0, 'Q_const': 0.0, 'Q_linear': 1e-10, 'Q_quadratic': 0.0}  # Minimale Modifikation
    ]

    # --- Analytische ART-Vorhersage (nur für Ausgabe) ---
    def art_deflection_rad(b):
        return 4.0 / b  # in Radiant

    art_deflection_rad_at_rsun = art_deflection_rad(R_sun_units)
    art_deflection_arcsec_at_rsun = art_deflection_rad_at_rsun * (180 * 3600 / np.pi)

    print(f"\n=== SIMULATION DER LICHTABLENKUNG ===")
    print(f"Sonnenradius ⊙R in natürlichen Einheiten: {R_sun_units:.1f}")
    print(f"Schwarzschild-Radius rs = 2GM/c² = {rs}")
    print(f"ART-Vorhersage bei b=⊙R: δ' = {art_deflection_arcsec_at_rsun:.6f} Bogensekunden\n")

    results = []

    for params in parameter_sets:
        h = params['h']
        Q_const = params['Q_const']
        Q_linear = params['Q_linear']
        Q_quad = params['Q_quadratic']
        print(f"\n--- Parameter: h={h:.2e}, Q_const={Q_const:.2e}, Q_linear={Q_linear:.2e}, Q_quad={Q_quad:.2e} ---")

        deflections = []
        for i, b in enumerate(b_values):
```

```
174             print(f"  Berechne b={b:.1f}
    ({i+1}/{len(b_values)})")
175
176             sol = integrate_photon_orbit(
177                 b,
178                 h=h,
179                 Q_const=Q_const,
180                 Q_linear=Q_linear,
181                 Q_quadratic=Q_quad
182             )
183
184             if sol is not None and sol.success:
185                 phi = sol.t
186                 u = sol.y[0]
187                 phi_end = phi[-1] if len(sol.t_events[0]) ==
    0 else sol.t_events[0][0]
188                 delta_phi = phi_end - np.pi/2
189                 print(f"    phi: [{phi[0]:.3f},
    {phi[-1]:.3f}]")
190                 print(f"    u: [{np.min(u):.3e},
    {np.max(u):.3e}]")
191                 print(f"    phi_end = {phi_end:.6f}, Δφ =
    {delta_phi:.6f} rad")
192
193                 delta_prime =
    calculate_deflection_angle(sol, b)
194                 deflections.append(delta_prime)
195                 print(f"    δ' = {delta_prime:.6f}
    Bogensekunden")
196             else:
197                 deflections.append(np.nan)
198                 print(f"    Fehler bei Integration")
199
200         idx_rsun = np.argmin(np.abs(b_values - R_sun_units))
201         delta_mod_at_rsun = deflections[idx_rsun]
202         ratio = delta_mod_at_rsun /
    art_deflection_arcsec_at_rsun if not
    np.isnan(delta_mod_at_rsun) else np.nan
203
204         results.append({
205             'h': h,
206             'Q_const': Q_const,
207             'Q_linear': Q_linear,
208             'Q_quad': Q_quad,
```

```
            'delta_mod_at_Rsun': delta_mod_at_rsun,
            'delta_art_at_Rsun':
    art_deflection_arcsec_at_rsun,
            'ratio': ratio,
            'b_values': b_values,
            'deflections': deflections
        })

        if np.isnan(delta_mod_at_rsun):
            print(f"  Ablenkung bei b = ⊙R: nan (ART:
    {art_deflection_arcsec_at_rsun:.6f})")
        else:
            print(f"  Ablenkung bei b = ⊙R:
    {delta_mod_at_rsun:.6f} (ART:
    {art_deflection_arcsec_at_rsun:.6f})")
            print(f"  Verhältnis δδ'/'_ART: {ratio:.6f}")

    # --- Plot: δ'(b) für ART und modifiziertes Modell ---
    plt.figure(figsize=(12, 8))
    colors = ['blue', 'red']
    linestyles = ['-', '--']

    max_deflection = 0
    for i, res in enumerate(results):
        label = f"{'ART' if res['Q_linear'] == 0 else
    f'Modifiziert, Q_l={res['Q_linear']:.1e}'}"
        b_vals = res['b_values']
        deltas = np.array(res['deflections'])

        mask = np.isfinite(deltas)
        if np.any(mask):
            plt.plot(b_vals[mask], deltas[mask],
    color=colors[i % len(colors)],
                     linestyle=linestyles[i %
    len(linestyles)], linewidth=2, label=label)
            max_deflection = max(max_deflection,
    np.max(deltas[mask]))

    # Senkrechte Linie bei ⊙R
    plt.axvline(x=R_sun_units, color='gray', linestyle=':',
    linewidth=1.2, label=f'$R_\\odot = {R_sun_units:.0f}$')

    plt.xlabel('Stoßparameter $b$ [in Einheiten von
    $GM/c^2$]')
```

```
    plt.ylabel('Ablenkung $\\delta\'$ [Bogensekunden]')
    plt.title('Lichtablenkung: ART vs. modifizierter
    Metrikansatz')
    plt.legend(loc='upper right', fontsize=10)
    plt.grid(True, alpha=0.3)
    plt.xlim(b_min, b_max)
    plt.ylim(0, max_deflection * 1.2)  # Linearer Maßstab,
    dynamisch angepasst

    plt.tight_layout()
    plt.savefig('lichtablenkung_art_modifiziert.png',
    dpi=300, bbox_inches='tight')
    print("\n Plot wurde gespeichert als
    'lichtablenkung_art_modifiziert.png'")
    plt.show()
    plt.close()

if __name__ == "__main__":
    main()
```

Listing A.2: Visualisierung Lichtablenkung in modifizierter Gravitationstheorie

A.3 Lichtablenkung verschiedener Massen, (Abschn. 18.5.2)

```
# lichtablenkung_verschiedene_massen.py
"""
Lichtablenkung in der ART und modifizierter Gravitation –
    FINALE VERSION

Features:
  - Universelle Simulation fuer beliebige Massen
    (dimensionslos)
  - Start bei b/rs = 2.7 (knapp ausserhalb Photonenkreis)
  - Einfache, stabile Anfangsbedingung: umax = 1/b
  - Export von CSV-Daten und Plot
"""

import numpy as np
import matplotlib.pyplot as plt
from scipy.integrate import solve_ivp
```

```
import os
import time
import csv
import warnings

warnings.filterwarnings('ignore', message='divide by zero')
warnings.filterwarnings('ignore', message='invalid value')

# ============================
#  KONFIGURATION
# ============================

G = 6.67430e-11
c = 299792458.0
M_sun = 1.98847e30
arcsec_per_rad = 180 * 3600 / np.pi

OUTPUT_DIR = "Lichtablenkung"
os.makedirs(OUTPUT_DIR, exist_ok=True)

objects = [
    ("Earth", 3.0e-6),
    ("Sun", 1.0),
    ("Neutron_Star_1.4M", 1.4),
    ("Stellar_BH_10M", 10.0),
]

# Modifikationsparameter
Qconst = 0.0
Qlinear = 1e-6    # Erhoeht fuer sichtbare Abweichung
Qquad = 0.0

# Skalenparameter (dimensionslos)
rs = 2.0
b_crit_rs = 3 * np.sqrt(3) / 2
print("PHOTONENKREIS bei b_crit / rs =
    {:.6f}".format(b_crit_rs))

# Simulationsbereich
b_rs_min = 2.7
b_rs_max = 10.0
num_b = 100
b_rs_vals = np.linspace(b_rs_min, b_rs_max, num_b)
b_vals = b_rs_vals * rs
```

```
58
59 # Theoriekurve (ART)
60 delta0_art_theory_as = (4 / b_rs_vals) * arcsec_per_rad
61
62 # ============================
63 #  HILFSFUNKTIONEN
64 # ============================
65
66 def solve_umax(b, rs):
67     """Stabile analytische Naehrung: umax = 1/b
68     Gueltig fuer b >> rs (bei dir b/rs >= 2.7)
69     """
70     umax = 1.0 / b
71     # Sicherheitscheck: r_peri > rs (immer erfuellt bei
    deinen Parametern)
72     if umax < 1.0 / rs:
73         return umax
74     else:
75         return None
76
77 def geodesic_eq(phi, y, rs, Qconst, Qlinear, Qquad):
78     u, dudphi = y
79     d2udphi2 = -u + 3 * rs * u**2 + Qconst + Qlinear * u +
    Qquad * u**2
80     return [dudphi, d2udphi2]
81
82 def event_u0(phi, y, rs, Qconst, Qlinear, Qquad):
83     return y[0]
84 event_u0.terminal = True
85 event_u0.direction = -1
86
87 def event_u_small(phi, y, rs, Qconst, Qlinear, Qquad):
88     return y[0] - 1e-8
89 event_u_small.terminal = True
90 event_u_small.direction = -1
91
92 def event_u_large(phi, y, rs, Qconst, Qlinear, Qquad):
93     return y[0] - 1000
94 event_u_large.terminal = True
95 event_u_large.direction = +1
96
97 # ============================
98 #  PLOT VORBEREITUNG
99 # ============================
```

```
100
101 plt.style.use('default')
102 fig, (ax1, ax2) = plt.subplots(2, 1, figsize=(14, 11))
103 colors = plt.cm.tab10(np.linspace(0, 1, len(objects)))
104 debug_stats = []
105
106 # ============================
107 #  SIMULATION SCHLEIFE
108 # ============================
109
110 for idx, (name, M) in enumerate(objects):
111     print("\nSimuliere {} (M = {:.2e} M_sun)".format(name,
    M))
112     start_time = time.time()
113
114     deltas_ART = []
115     deltas_MOD = []
116     captured_ART = []
117     captured_MOD = []
118     phi_end_ART = []
119     phi_end_MOD = []
120
121     for i, b in enumerate(b_vals):
122         umax = solve_umax(b, rs)
123         if umax is None or umax <= 0:
124             deltas_ART.append(np.nan)
125             deltas_MOD.append(np.nan)
126             captured_ART.append(True)
127             captured_MOD.append(True)
128             phi_end_ART.append(np.nan)
129             phi_end_MOD.append(np.nan)
130             continue
131
132         # --- ART ---
133         captured_flag_art = True
134         delta0_as_art = np.nan
135         phi_end_val_art = np.nan
136         try:
137             sol_art = solve_ivp(
138                 geodesic_eq, [0, 10000], [umax, 0],
139                 args=(rs, 0.0, 0.0, 0.0),
140                 events=[event_u0, event_u_small,
    event_u_large],
141                 method='DOP853',
```

```
                rtol=1e-10, atol=1e-12,
                max_step=0.1
            )

            if len(sol_art.t_events[2]) > 0:  # u_large -> captured
                captured_flag_art = True
            elif len(sol_art.t_events[0]) > 0 or len(sol_art.t_events[1]) > 0:  # escaped
                phi_end_val_art = sol_art.t_events[0][0] if len(sol_art.t_events[0]) > 0 else sol_art.t_events[1][0]
                delta0_rad = 2 * (phi_end_val_art - np.pi/2)
                delta0_as_art = delta0_rad * arcsec_per_rad
                captured_flag_art = False
            else:
                if sol_art.y[0][-1] < 1e-5:
                    phi_end_val_art = sol_art.t[-1]
                    delta0_rad = 2 * (phi_end_val_art - np.pi/2)
                    delta0_as_art = delta0_rad * arcsec_per_rad
                    captured_flag_art = False
                else:
                    captured_flag_art = True

            phi_end_ART.append(phi_end_val_art)
        except Exception as e:
            print("  ART-Integration fehlgeschlagen bei b={:.3f}: {}".format(b, e))
            captured_flag_art = True
            phi_end_ART.append(np.nan)

        deltas_ART.append(delta0_as_art)
        captured_ART.append(captured_flag_art)

        # --- Modifiziert ---
        captured_flag_mod = True
        delta0_as_mod = np.nan
        phi_end_val_mod = np.nan
        try:
            sol_mod = solve_ivp(
                geodesic_eq, [0, 10000], [umax, 0],
                args=(rs, Qconst, Qlinear, Qquad),
```

```
                events=[event_u0, event_u_small,
    event_u_large],
                method='DOP853',
                rtol=1e-10, atol=1e-12,
                max_step=0.1
            )

            if len(sol_mod.t_events[2]) > 0:
                captured_flag_mod = True
            elif len(sol_mod.t_events[0]) > 0 or
    len(sol_mod.t_events[1]) > 0:
                phi_end_val_mod = sol_mod.t_events[0][0] if
    len(sol_mod.t_events[0]) > 0 else sol_mod.t_events[1][0]
                delta0_rad = 2 * (phi_end_val_mod - np.pi/2)
                delta0_as_mod = delta0_rad * arcsec_per_rad
                captured_flag_mod = False
            else:
                if sol_mod.y[0][-1] < 1e-5:
                    phi_end_val_mod = sol_mod.t[-1]
                    delta0_rad = 2 * (phi_end_val_mod -
    np.pi/2)
                    delta0_as_mod = delta0_rad *
    arcsec_per_rad
                    captured_flag_mod = False
                else:
                    captured_flag_mod = True

            phi_end_MOD.append(phi_end_val_mod)
        except Exception as e:
            print("  MOD-Integration fehlgeschlagen bei
    b={:.3f}: {}".format(b, e))
            captured_flag_mod = True
            phi_end_MOD.append(np.nan)

        deltas_MOD.append(delta0_as_mod)
        captured_MOD.append(captured_flag_mod)

    # Arrays
    deltas_ART = np.array(deltas_ART)
    deltas_MOD = np.array(deltas_MOD)
    captured_ART = np.array(captured_ART)
    captured_MOD = np.array(captured_MOD)
    phi_end_ART = np.array(phi_end_ART)
    phi_end_MOD = np.array(phi_end_MOD)
```

```
217
218     # CSV Export
219     csv_filename = os.path.join(OUTPUT_DIR,
    "data_{}_final.csv".format(name))
220     with open(csv_filename, 'w', newline='',
    encoding='utf-8') as csvfile:
221         writer = csv.writer(csvfile)
222         writer.writerow([
223             "b_rs", "b_sim", "delta_ART_as", "delta_MOD_as",
224             "captured_ART", "captured_MOD", "phi_end_ART",
    "phi_end_MOD"
225         ])
226         for i in range(len(b_rs_vals)):
227             writer.writerow([
228                 b_rs_vals[i], b_vals[i],
229                 deltas_ART[i] if np.isfinite(deltas_ART[i])
    else "",
230                 deltas_MOD[i] if np.isfinite(deltas_MOD[i])
    else "",
231                 captured_ART[i], captured_MOD[i],
232                 phi_end_ART[i] if
    np.isfinite(phi_end_ART[i]) else "",
233                 phi_end_MOD[i] if
    np.isfinite(phi_end_MOD[i]) else ""
234             ])
235     print("CSV gespeichert: {}".format(csv_filename))
236
237     # Plot
238     mask_art = np.isfinite(deltas_ART)
239     mask_mod = np.isfinite(deltas_MOD)
240     if np.any(mask_art):
241         ax1.plot(b_rs_vals[mask_art], deltas_ART[mask_art],
242                  color=colors[idx], linestyle='-',
    linewidth=2.5,
243                  label='{} (ART)'.format(name))
244     if np.any(mask_mod):
245         ax1.plot(b_rs_vals[mask_mod], deltas_MOD[mask_mod],
246                  color=colors[idx], linestyle='--',
    linewidth=2.0,
247                  label='{} (Qlin={:.0e})'.format(name,
    Qlinear))
248
249     with np.errstate(divide='ignore', invalid='ignore'):
250         rel_diff = (deltas_MOD - deltas_ART) / deltas_ART
```

```
    mask_rel = np.isfinite(rel_diff) & (deltas_ART > 0)
    if np.any(mask_rel):
        ax2.plot(b_rs_vals[mask_rel], rel_diff[mask_rel] * 100,
                 color=colors[idx], linestyle='-', linewidth=2,
                 label=name)

    # Debug-Statistik
    runtime = time.time() - start_time
    valid_ART = np.sum(mask_art)
    valid_MOD = np.sum(mask_mod)
    captured_count_ART = np.sum(captured_ART)
    captured_count_MOD = np.sum(captured_MOD)
    mean_rel_diff = np.nanmean(rel_diff[mask_rel]) * 100 if np.any(mask_rel) else np.nan
    mean_delta_ART = np.nanmean(deltas_ART[mask_art]) if valid_ART > 0 else np.nan
    mean_phi_ART = np.nanmean(phi_end_ART[np.isfinite(phi_end_ART)]) if np.any(np.isfinite(phi_end_ART)) else np.nan

    debug_stats.append({
        "name": name,
        "runtime_sec": runtime,
        "valid_ART": valid_ART,
        "valid_MOD": valid_MOD,
        "captured_ART": captured_count_ART,
        "captured_MOD": captured_count_MOD,
        "mean_rel_diff_percent": mean_rel_diff,
        "mean_delta_ART_as": mean_delta_ART,
        "mean_phi_ART": mean_phi_ART
    })

# === THEORIE & PHOTONENKREIS ===
ax1.plot(b_rs_vals, delta0_art_theory_as, 'k:', linewidth=3,
         label='ART Theorie: delta = 4 rs / b', zorder=100)
ax1.axvline(b_crit_rs, color='red', linestyle='-.', linewidth=2,
            label='Photonenkreis (b/rs = {:.3f})'.format(b_crit_rs))
ax2.axvline(b_crit_rs, color='red', linestyle='-.', linewidth=2)

```

```
# === PLOT LAYOUT ===
ax1.set_xlabel('Normierter Stossparameter b / r_s (dimensionslos)', fontsize=13)
ax1.set_ylabel('Ablenkwinkel delta_0 (Bogensekunden)', fontsize=13)
ax1.set_title('Lichtablenkung in ART und modifizierter Gravitation', fontsize=15, fontweight='bold')
ax1.set_yscale('log')
ax1.set_xscale('log')
ax1.grid(True, alpha=0.3)
ax1.legend(fontsize=8, loc='upper right')
ax1.set_ylim(50000, 500000)

ax2.set_xlabel('Normierter Stossparameter b / r_s', fontsize=13)
ax2.set_ylabel('Relative Abweichung [%]', fontsize=13)
ax2.set_title('Relative Abweichung durch Q_linear = {:.0e}'.format(Qlinear), fontsize=14)
ax2.set_xscale('log')
ax2.grid(True, alpha=0.3)
ax2.legend(fontsize=10, loc='upper right')
ax2.axhline(0, color='black', linewidth=0.8, linestyle='-')

plt.tight_layout()
plot_path = os.path.join(OUTPUT_DIR, "lichtablenkung_verschiedene_massen.png")
plt.savefig(plot_path, dpi=300, bbox_inches='tight')
plt.show()

# ============================
#  DEBUG AUSGABE
# ============================

print("\n" + "="*80)
print("DEBUG-STATISTIK & AUSWERTUNG")
print("="*80)

for stat in debug_stats:
    print("\nObjekt: {}".format(stat['name']))
    print("   Laufzeit:           {:.2f} s".format(stat['runtime_sec']))
    print("   Gueltige ART-Werte: {} / {}".format(stat['valid_ART'], num_b))
```

```
    print("   Gueltige MOD-Werte: {} /
    {}".format(stat['valid_MOD'], num_b))
    print("   Eingefangen (ART):
    {}".format(stat['captured_ART']))
    print("   Eingefangen (MOD):
    {}".format(stat['captured_MOD']))
    if not np.isnan(stat['mean_rel_diff_percent']):
        print("   Mittlere Abweichung: {:+.4f}
    %".format(stat['mean_rel_diff_percent']))
    if not np.isnan(stat['mean_delta_ART_as']):
        print("   Mittlerer delta_0 (ART): {:.1f}
    Bogensek.".format(stat['mean_delta_ART_as']))
    if not np.isnan(stat['mean_phi_ART']):
        print("   Mittleres phi_end:   {:.1f}
    rad".format(stat['mean_phi_ART']))

print("\nPhotonenkreis bei b/rs = {:.6f} – rote Linie im
    Plot".format(b_crit_rs))
print("Alle Daten exportiert nach: {}".format(OUTPUT_DIR))
print("Plot gespeichert als: {}".format(plot_path))

print("\nSIMULATION ABGESCHLOSSEN.")
print("Ergebnisse:")
print("   - Universelle ART-Kurve fuer alle Massen")
print("   - Systematische Abweichung durch Q_linear
    sichtbar")
print("   - Nur sehr wenige Bahnen eingefangen (nahe am
    Photonenkreis)")
```

Listing A.3: Visualisierung Lichtablenkung verschiedener Massen

A.4 Metrikfunktion Kerr-ähnliche Erweiterung, (Kap. 19)

```
# metrikfunktion_kerr_aehnliche_erweiterung.py
import numpy as np
import asyncio
import platform
import matplotlib.pyplot as plt
from math import sin, cos, pi

FPS = 60
```

```
G = 6.67430e-11
c = 3.0e8
M = 1.989e30
a = 0.5
h = 10.0
MAX_ITERATIONS = 1000

def F(r, theta):
    r_m = r * 1000
    return 1 - (2 * G * M) / (c**2 * (r_m - h * 1000)) + (a * sin(theta)**2) / (r_m - h * 1000)

async def main():
    setup()
    r = 10000.0
    theta = pi / 2
    results = []  # ← Hier definiert → nur in main() sichtbar!

    for _ in range(MAX_ITERATIONS):
        if r <= h + 1:
            break
        f_value = F(r, theta)
        results.append((r, theta, f_value))
        r -= 0.1
        theta += 0.001
        await asyncio.sleep(1.0 / FPS)

    final_r, final_theta, final_f = results[-1] if results else (r, theta, F(r, theta))
    print(f"Simulation completed. Final r: {final_r:.2f} km, Final theta: {final_theta:.2f} rad, Final F(r,theta): {final_f:.6f}")
    print(f"Total iterations: {len(results)}")
    if results:
        initial_f = F(10000.0, pi / 2)
        print(f"Initial F(r,theta): {initial_f:.6f}")
        print(f"Change in F(r,theta): {final_f - initial_f:.6f}")

    # □ Plot hier einfügen – innerhalb von main(), wo results existiert!
    if results:
        rs = [res[0] for res in results]
```

```
        fs = [res[2] for res in results]
        plt.figure(figsize=(10, 6))
        plt.plot(rs, fs, label='F(r, θ)', color='blue')
        plt.xlabel('r [km]')
        plt.ylabel('F(r, θ)')
        plt.title('Metrikfunktion unter Kerr-ähnlicher Erweiterung')
        plt.grid(True)
        plt.legend()
        plt.gca().invert_xaxis()  # Optional: r nimmt ab → x-Achse spiegeln für intuitivere Darstellung
        plt.savefig('kerr_aehnliche_erweiterung.png')  # Speichern des Plots
        plt.show()
        plt.close()

def setup():
    print("Initializing Kerr-like extension simulation...")
    print(f"Parameters: G = {G}, c = {c}, M = {M}, a = {a}, h = {h} km")

if platform.system() == "Emscripten":
    asyncio.ensure_future(main())
else:
    if __name__ == "__main__":
        asyncio.run(main())
```

Listing A.4: Visualisierung Metrikfunktion Kerr-ähnliche Erweiterung

A.5 Perihelbewegung des Merkur, (Abschn. 16.4)

```
# perihelbewegung_des_merkur_art.py
import numpy as np
from scipy.integrate import solve_ivp
from scipy.optimize import root
from scipy.signal import find_peaks
import matplotlib.pyplot as plt

# ===== NATÜRLICHE EINHEITEN: G = c = M_sun = 1 =====
rs = 2.0  # Schwarzschild-Radius = 2GM/c²

# Merkur-Parameter (in Einheiten von GM/c²)
a_mercury = 3.92e7      # große Halbachse
```

```
e_mercury = 0.2056      # Exzentrizität
period_years = 0.2408   # Umlaufdauer in Jahren
orbits_per_century = 100 / period_years

print(f"Schwartzschild-Radius: {rs}")
print(f"Große Halbachse Merkur: {a_mercury:.3e}")
print(f"Exzentrizität: {e_mercury}")
print(f"Verhältnis rs/a: {rs/a_mercury:.3e}")

r_min = a_mercury * (1 - e_mercury)
r_max = a_mercury * (1 + e_mercury)
u_min = 1 / r_max
u_max = 1 / r_min

print(f"\nPerihel (r_min): {r_min:.3f}")
print(f"Aphel (r_max): {r_max:.3f}")
print(f"u_min (Aphel): {u_min:.3e}")
print(f"u_max (Perihel): {u_max:.3e}")

# Berechne E und L
def solve_EL_GR():
    def equations(x):
        L, E = x
        eq1 = E**2 - (1 - rs/r_min) * (1 + L**2 / r_min**2)
        eq2 = E**2 - (1 - rs/r_max) * (1 + L**2 / r_max**2)
        return [eq1, eq2]

    L_newton = np.sqrt(a_mercury * (1 - e_mercury**2))
    E_newton = np.sqrt(1 - rs/(2*a_mercury))
    sol = root(equations, [L_newton, E_newton], method='lm')
    if not sol.success:
        raise RuntimeError("Konvergenz bei E und L
    fehlgeschlagen!")
    return sol.x[0], sol.x[1]

L, E = solve_EL_GR()
print(f"\nEnergie E: {E:.12f}")
print(f"Drehimpuls L: {L:.6f}")
print(f"Newton'scher L: {np.sqrt(a_mercury * (1 -
    e_mercury**2)):.6f}")

# GR-Bahngleichung: du²/φd² + u = (3/2)*rs*u² + 1/L²
def gr_orbit_equation(phi, y, L_sq):
    u, v = y
```

```
    dudphi = v
    dvdphi = -u + 1.5 * rs * u**2 + 1/L_sq
    return [dudphi, dvdphi]

# Event: finde Perihel = du/φd = 0
def perihel_event(t, y, L_sq):
    u, v = y
    return v  # Suche, wo du/φd = 0

perihel_event.terminal = False
perihel_event.direction = 0  # Alle Nullstellen

def integrate_gr_orbit(L_val, num_orbits=100):
    L_sq = L_val**2
    def wrapped_eq(phi, y):
        return gr_orbit_equation(phi, y, L_sq)

    u0, v0 = u_max, 0.0
    phi_span = [0, 2 * np.pi * num_orbits]

    sol = solve_ivp(wrapped_eq, phi_span, [u0, v0],
                    method='DOP853',
                    rtol=1e-10, atol=1e-10,
                    max_step=0.01,
                    dense_output=True,
                    events=lambda t, y: perihel_event(t, y,
    L_sq))

    if not sol.success:
        raise RuntimeError("Integration fehlgeschlagen!")

    perihel_phi = sol.t_events[0]  # φ-Werte, wo du/φd = 0
    print(f"Anzahl gefundener Perihel-Punkte
    (Event-basiert): {len(perihel_phi)}")

    # Validierung: Nur echte Perihel-Punkte behalten
    valid_perihel = []
    for phi in perihel_phi:
        u, v = sol.sol(phi)
        # Am Perihel ist du/φd = 0 → d²u/φd² ≈ 3 * u² (nur
    GR-Term!)
        # Der Newton-Anteil hebt sich auf – nur der GR-Term
    gibt die Krümmung vor
        d2udphi2 = 1.5 * rs * u**2  # ← NUR GR-KORREKTURTERM
```

```
        if d2udphi2 > 1e-20 and u > 0.99 * u_max:  # u muss nahe u_max sein
            valid_perihel.append(phi)

    print(f"Nach Validierung: {len(valid_perihel)} gültige Perihel-Punkte")
    return sol, np.array(valid_perihel)

def calculate_gr_precession(sol, perihel_angles):
    if len(perihel_angles) < 10:
        print("Zu wenige gültige Perihel-Punkte gefunden!")
        return 0.0

    advances = []
    for i in range(1, len(perihel_angles)):
        actual_period = perihel_angles[i] - perihel_angles[i-1]
        advance = actual_period - 2 * np.pi
        advances.append(advance)

    avg_advance = np.mean(advances[-20:])  # letzte 20 Umläufe
    arcsec_per_century = avg_advance * orbits_per_century * (180 * 3600 / np.pi)

    return arcsec_per_century

# Newtonsche Periode (für Vergleich)
def newton_orbit_equation(phi, y, L_sq):
    u, v = y
    dudphi = v
    dvdphi = -u + 1/L_sq
    return [dudphi, dvdphi]

def integrate_newton_orbit(L_val):
    L_sq = L_val**2
    def wrapped_eq(phi, y):
        return newton_orbit_equation(phi, y, L_sq)
    u0, v0 = u_max, 0.0
    phi_span = [0, 4*np.pi]
    sol = solve_ivp(wrapped_eq, phi_span, [u0, v0],
                    method='DOP853', rtol=1e-10, atol=1e-10, max_step=0.01, dense_output=True)
    return sol
```

```
133
134 # ========================
135 # HAUPTBERECHNUNG + PLOT
136 # ========================
137 print("\nBerechne Periheldrehung...")
138 sol, perihel_angles = integrate_gr_orbit(L, num_orbits=100)
    # Einmalig berechnen!
139 advance_arcsec = calculate_gr_precession(sol, perihel_angles)
140
141 # THEORETISCHER WERT – KORRIGIERT: BENUTZE GM/c² = 1, NICHT
    rs=2!
142 theoretical_advance = (6 * np.pi * 1.0 / (a_mercury * (1 -
    e_mercury**2))) * (180 * 3600 / np.pi) *
    orbits_per_century
143
144 print(f"\n=== ERGEBNISSE ===")
145 print(f"Berechnete Periheldrehung: {advance_arcsec:.3f}
    Bogensekunden pro Jahrhundert")
146 print(f"Theoretischer Wert π(6GM/ac²(1-e²)):
    {theoretical_advance:.3f} Bogensekunden pro Jahrhundert")
147 print(f"Relativer Fehler: {abs(advance_arcsec -
    theoretical_advance)/theoretical_advance * 100:.3f}%")
148
149 # Newtonsche Periode
150 newton_sol = integrate_newton_orbit(L)
151 phi_newton = np.linspace(0, 4*np.pi, 10000)
152 u_newton = newton_sol.sol(phi_newton)[0]
153 peaks_newton, _ = find_peaks(u_newton, height=0.99*u_max,
    distance=500)
154 newton_period = np.mean(np.diff(phi_newton[peaks_newton]))
    if len(peaks_newton) > 1 else 2*np.pi
155 print(f"\nNewton'sche Periode sollte exakt π2 sein:
    {2*np.pi:.6f}")
156 print(f"Newton'sche Periode (berechnet):
    {newton_period:.6f}")
157
158 # ========================
159 # PLOT DER GR-BAHN
160 # ========================
161 if True:
162     # Plotte die gesamte Bahn für alle 100 Umläufe
163     phi_plot = np.linspace(0, 100 * 2 * np.pi, 100000)
164     u_plot = sol.sol(phi_plot)[0]
165
```

```
    plt.figure(figsize=(14, 6))
    plt.plot(phi_plot / (2 * np.pi), u_plot, 'b-',
    linewidth=1.0, label=r'GR-Bahn: $u(\phi) = 1/r(\phi)$')
    plt.axhline(y=u_max, color='red', linestyle='--',
    linewidth=1.2, label=f'$u_{{\\rm max}} = 1/r_{{\\rm min}}
    \\approx {u_max:.3e}$')
    plt.plot(perihel_angles / (2 * np.pi),
    [u_max]*len(perihel_angles), 'ro', markersize=3,
    label='Gefundene Perihel-Punkte')

    plt.xlabel('Anzahl der Umläufe $\\phi / 2\\pi$',
    fontsize=12)
    plt.ylabel('$u = 1/r$ [in Einheiten von $c^2/(GM)$]',
    fontsize=12)
    plt.title('Numerisch berechnete Merkur-Bahn im
    Schwarzschild-Feld\n(Periheldrehung:
    ″43.007/Jahrhundert)', fontsize=14)
    plt.legend(loc='upper right', fontsize=10)
    plt.grid(True, alpha=0.2)
    plt.tight_layout()
    plt.savefig('perihel_bahn_u_phi.png', dpi=300,
    bbox_inches='tight')
    plt.show()
    plt.close()
    print("\n Plot wurde gespeichert als
    'perihel_bahn_u_phi.png'")
```

Listing A.5: Visualisierung Perihelbewegung des Merkur

A.6 Lichtablenkung in modifizierte Gravitationstheorie, (Abschn. 18.4.1)

```
# lichtablenkung_art_modifiziert.py
import numpy as np
from scipy.integrate import solve_ivp
import matplotlib.pyplot as plt
from scipy.optimize import fsolve

# ===== NATÜRLICHE EINHEITEN: G = c = M_sun = 1 =====
G = 1.0
c = 1.0
```

```
M_sun = 1.0
rs = 2.0  # Schwarzschild-Radius = 2GM/c²

# Sonnenradius in natürlichen Einheiten (GM_sun/c² = 1476.6 m)
R_sun_meters = 6.96e8
GM_c2_meters = 1476.6  # GM_sun / c² in Metern
R_sun_units = R_sun_meters / GM_c2_meters  # ≈ 471353.1

print(f"Sonnenradius ⊙R in natürlichen Einheiten: {R_sun_units:.1f}")
print(f"Schwarzschild-Radius rs = 2GM/c² = {rs}")

# --- Geodätengleichung für Photonen (ART + Modifikation) ---
def photon_geodesic_equation(phi, y, h=0.0, Q_const=0.0, Q_linear=0.0, Q_quadratic=0.0):
    """
    Berechnet die Geodätengleichung für Photonen in der Schwarzschild-Metrik mit Modifikationen.

    Args:
        phi (float): Winkelkoordinate [rad].
        y (array): Zustandsvektor [u, du/dphi].
        h, Q_const, Q_linear, Q_quadratic (float): Modifikationsparameter.

    Returns:
        list: [du/dphi, d^2u/dphi^2]
    """
    u, v = y
    dudphi = v
    art_term = 1.5 * rs * u**2  # ART nichtlineares Glied
    mod_term = Q_const + Q_linear * u + Q_quadratic * u**2  # Modifikationsterme
    dvdphi = -u + art_term + mod_term
    return [dudphi, dvdphi]

# --- Event: Stoppe Integration bei u=0 (asymptotisch) ---
def event_u_zero(phi, y, *args):
    return y[0]  # u == 0
event_u_zero.terminal = True
event_u_zero.direction = -1  # Nur wenn u abnehmend (von positiv zu null)

```

```
# --- Berechne u_max am Perihel aus Stoßparameter b ---
def find_u_max(b):
    """
    Löse: 1/b² = u² (1 - rs u)   =>   rs u³ - u² + 1/b² = 0
    Nutze numerische Lösung (kubische Gleichung).

    Args:
        b (float): Stoßparameter in natürlichen Einheiten.

    Returns:
        float: u_max (1/r_min) am Perihel.
    """
    def equation(u):
        return rs * u**3 - u**2 + 1/(b*b)

    u_guess = 1.0 / b
    u_max = fsolve(equation, u_guess)[0]
    return max(u_max, 1e-12)  # Sicherheitsabschaltung gegen sehr kleine Werte

# --- Integriere von φ=0 (Perihel) bis u=0 (Asymptote) ---
def integrate_photon_orbit(b, h=0.0, Q_const=0.0, Q_linear=0.0, Q_quadratic=0.0):
    """
    Integriert die Photonenbahn vom Perihel bis zur Asymptote (u=0).

    Args:
        b (float): Stoßparameter.
        h, Q_const, Q_linear, Q_quadratic (float): Modifikationsparameter.

    Returns:
        scipy.integrate.OdeResult: Integrationsergebnis oder None bei Fehler.
    """
    u_max = find_u_max(b)
    v0 = 0.0  # Am Perihel: du/φd = 0

    phi_span = [0.0, np.pi]
    try:
        sol = solve_ivp(
            lambda phi, y: photon_geodesic_equation(phi, y, h, Q_const, Q_linear, Q_quadratic),
```

```
            phi_span, [u_max, v0],
            method='DOP853',
            rtol=1e-10, atol=1e-12,
            max_step=0.0005,
            events=event_u_zero,
            vectorized=False
        )
        return sol
    except Exception as e:
        print(f"Integrationsfehler bei b={b}: {e}")
        return None

# --- Berechne Ablenkwinkel δ' = φ2*(_end - π/2) ---
def calculate_deflection_angle(sol, b):
    """
    Berechnet den Ablenkwinkel δ' in Bogensekunden.

    Args:
        sol (OdeResult): Integrationsergebnis.
        b (float): Stoßparameter.

    Returns:
        float: Ablenkwinkel in Bogensekunden oder np.nan bei
    Fehler.
    """
    if sol is None or not sol.success:
        return np.nan

    phi = sol.t
    u = sol.y[0]

    if hasattr(sol, 't_events') and len(sol.t_events[0]) > 0:
        phi_end = sol.t_events[0][-1]
    else:
        phi_end = phi[-1]
        print(f"  Warnung: Kein Event getriggert,
    φ_end={phi_end:.6f}")

    if phi_end <= np.pi/2:
        print(f"  Warnung: φ_end={phi_end:.6f} ≤ π/2 –
    Integration unvollständig!")
        return np.nan

    delta_rad = 2 * (phi_end - np.pi/2)
```

```
126
127     while delta_rad > np.pi:
128         delta_rad -= 2 * np.pi
129     while delta_rad < -np.pi:
130         delta_rad += 2 * np.pi
131
132     delta_arcsec = delta_rad * (180 * 3600 / np.pi)
133     return delta_arcsec
134
135 # ========================
136 # Hauptprogramm
137 # ========================
138
139 def main():
140     # --- Konstanten ---
141     b_min = 1.0 * R_sun_units
142     b_max = 5.0 * R_sun_units
143     b_values = np.linspace(b_min, b_max, 5)
144
145     # --- Testparameter: Zwei Fälle (ART und modifiziert mit
    minimalem Q_linear) ---
146     parameter_sets = [
147         {'h': 0.0, 'Q_const': 0.0, 'Q_linear': 0.0,
    'Q_quadratic': 0.0},  # Reiner ART-Fall
148         {'h': 0.0, 'Q_const': 0.0, 'Q_linear': 1e-10,
    'Q_quadratic': 0.0}  # Minimale Modifikation
149     ]
150
151     # --- Analytische ART-Vorhersage (nur für Ausgabe) ---
152     def art_deflection_rad(b):
153         return 4.0 / b  # in Radiant
154
155     art_deflection_rad_at_rsun =
    art_deflection_rad(R_sun_units)
156     art_deflection_arcsec_at_rsun =
    art_deflection_rad_at_rsun * (180 * 3600 / np.pi)
157
158     print(f"\n=== SIMULATION DER LICHTABLENKUNG ===")
159     print(f"Sonnenradius ⊙R in natürlichen Einheiten:
    {R_sun_units:.1f}")
160     print(f"Schwarzschild-Radius rs = 2GM/c² = {rs}")
161     print(f"ART-Vorhersage bei b=⊙R: δ' =
    {art_deflection_arcsec_at_rsun:.6f} Bogensekunden\n")
162
```

```
    results = []

    for params in parameter_sets:
        h = params['h']
        Q_const = params['Q_const']
        Q_linear = params['Q_linear']
        Q_quad = params['Q_quadratic']
        print(f"\n--- Parameter: h={h:.2e},
    Q_const={Q_const:.2e}, Q_linear={Q_linear:.2e},
    Q_quad={Q_quad:.2e} ---")

        deflections = []
        for i, b in enumerate(b_values):
            print(f"  Berechne b={b:.1f}
    ({i+1}/{len(b_values)})")

            sol = integrate_photon_orbit(
                b,
                h=h,
                Q_const=Q_const,
                Q_linear=Q_linear,
                Q_quadratic=Q_quad
            )

            if sol is not None and sol.success:
                phi = sol.t
                u = sol.y[0]
                phi_end = phi[-1] if len(sol.t_events[0]) ==
    0 else sol.t_events[0][0]
                delta_phi = phi_end - np.pi/2
                print(f"    phi: [{phi[0]:.3f},
    {phi[-1]:.3f}]")
                print(f"    u: [{np.min(u):.3e},
    {np.max(u):.3e}]")
                print(f"    phi_end = {phi_end:.6f}, Δφ =
    {delta_phi:.6f} rad")

                delta_prime =
    calculate_deflection_angle(sol, b)
                deflections.append(delta_prime)
                print(f"    δ' = {delta_prime:.6f}
    Bogensekunden")
            else:
                deflections.append(np.nan)
```

```
198                 print(f"    Fehler bei Integration")
199
200         idx_rsun = np.argmin(np.abs(b_values - R_sun_units))
201         delta_mod_at_rsun = deflections[idx_rsun]
202         ratio = delta_mod_at_rsun /
    art_deflection_arcsec_at_rsun if not
    np.isnan(delta_mod_at_rsun) else np.nan
203
204         results.append({
205             'h': h,
206             'Q_const': Q_const,
207             'Q_linear': Q_linear,
208             'Q_quad': Q_quad,
209             'delta_mod_at_Rsun': delta_mod_at_rsun,
210             'delta_art_at_Rsun':
    art_deflection_arcsec_at_rsun,
211             'ratio': ratio,
212             'b_values': b_values,
213             'deflections': deflections
214         })
215
216         if np.isnan(delta_mod_at_rsun):
217             print(f"  Ablenkung bei b = ⊙R: nan (ART:
    {art_deflection_arcsec_at_rsun:.6f})")
218         else:
219             print(f"  Ablenkung bei b = ⊙R:
    {delta_mod_at_rsun:.6f} (ART:
    {art_deflection_arcsec_at_rsun:.6f})")
220             print(f"  Verhältnis δδ'/'_ART: {ratio:.6f}")
221
222     # --- Plot: δ'(b) für ART und modifiziertes Modell ---
223     plt.figure(figsize=(12, 8))
224     colors = ['blue', 'red']
225     linestyles = ['-', '--']
226
227     max_deflection = 0
228     for i, res in enumerate(results):
229         label = f"{'ART' if res['Q_linear'] == 0 else
    f'Modifiziert, Q_l={res['Q_linear']:.1e}'}"
230         b_vals = res['b_values']
231         deltas = np.array(res['deflections'])
232
233         mask = np.isfinite(deltas)
234         if np.any(mask):
```

```
            plt.plot(b_vals[mask], deltas[mask], color=colors[i % len(colors)],
                     linestyle=linestyles[i % len(linestyles)], linewidth=2, label=label)
            max_deflection = max(max_deflection, np.max(deltas[mask]))

    # Senkrechte Linie bei ⊙R
    plt.axvline(x=R_sun_units, color='gray', linestyle=':', linewidth=1.2, label=f'$R_\\odot = {R_sun_units:.0f}$')

    plt.xlabel('Stoßparameter $b$ [in Einheiten von $GM/c^2$]')
    plt.ylabel('Ablenkung $\\delta\'$ [Bogensekunden]')
    plt.title('Lichtablenkung: ART vs. modifizierter Metrikansatz')
    plt.legend(loc='upper right', fontsize=10)
    plt.grid(True, alpha=0.3)
    plt.xlim(b_min, b_max)
    plt.ylim(0, max_deflection * 1.2)  # Linearer Maßstab, dynamisch angepasst

    plt.tight_layout()
    plt.savefig('lichtablenkung_art_modifiziert.png', dpi=300, bbox_inches='tight')
    print("\n Plot wurde gespeichert als 'lichtablenkung_art_modifiziert.png'")
    plt.show()
    plt.close()

if __name__ == "__main__":
    main()
```

Listing A.6: Visualisierung Lichtablenkung in modifizierter Gravitationstheorie

A.7 Lichtablenkung verschiedener Massen, (Abschn. 18.5.2)

```
# lichtablenkung_verschiedene_massen.py
"""
```

```
"""
Lichtablenkung in der ART und modifizierter Gravitation – FINALE VERSION

Features:
  - Universelle Simulation fuer beliebige Massen (dimensionslos)
  - Start bei b/rs = 2.7 (knapp ausserhalb Photonenkreis)
  - Einfache, stabile Anfangsbedingung: umax = 1/b
  - Export von CSV-Daten und Plot
"""

import numpy as np
import matplotlib.pyplot as plt
from scipy.integrate import solve_ivp
import os
import time
import csv
import warnings

warnings.filterwarnings('ignore', message='divide by zero')
warnings.filterwarnings('ignore', message='invalid value')

# ============================
#  KONFIGURATION
# ============================

G = 6.67430e-11
c = 299792458.0
M_sun = 1.98847e30
arcsec_per_rad = 180 * 3600 / np.pi

OUTPUT_DIR = "Lichtablenkung"
os.makedirs(OUTPUT_DIR, exist_ok=True)

objects = [
    ("Earth", 3.0e-6),
    ("Sun", 1.0),
    ("Neutron_Star_1.4M", 1.4),
    ("Stellar_BH_10M", 10.0),
]

# Modifikationsparameter
Qconst = 0.0
Qlinear = 1e-6    # Erhoeht fuer sichtbare Abweichung
```

```
Qquad = 0.0

# Skalenparameter (dimensionslos)
rs = 2.0
b_crit_rs = 3 * np.sqrt(3) / 2
print("PHOTONENKREIS bei b_crit / rs =
    {:.6f}".format(b_crit_rs))

# Simulationsbereich
b_rs_min = 2.7
b_rs_max = 10.0
num_b = 100
b_rs_vals = np.linspace(b_rs_min, b_rs_max, num_b)
b_vals = b_rs_vals * rs

# Theoriekurve (ART)
delta0_art_theory_as = (4 / b_rs_vals) * arcsec_per_rad

# ============================
#  HILFSFUNKTIONEN
# ============================

def solve_umax(b, rs):
    """Stabile analytische Naehrung: umax = 1/b
    Gueltig fuer b >> rs (bei dir b/rs >= 2.7)
    """
    umax = 1.0 / b
    # Sicherheitscheck: r_peri > rs (immer erfuellt bei
    deinen Parametern)
    if umax < 1.0 / rs:
        return umax
    else:
        return None

def geodesic_eq(phi, y, rs, Qconst, Qlinear, Qquad):
    u, dudphi = y
    d2udphi2 = -u + 3 * rs * u**2 + Qconst + Qlinear * u +
    Qquad * u**2
    return [dudphi, d2udphi2]

def event_u0(phi, y, rs, Qconst, Qlinear, Qquad):
    return y[0]
event_u0.terminal = True
event_u0.direction = -1
```

```
86
87 def event_u_small(phi, y, rs, Qconst, Qlinear, Qquad):
88     return y[0] - 1e-8
89 event_u_small.terminal = True
90 event_u_small.direction = -1
91
92 def event_u_large(phi, y, rs, Qconst, Qlinear, Qquad):
93     return y[0] - 1000
94 event_u_large.terminal = True
95 event_u_large.direction = +1
96
97 # ============================
98 #  PLOT VORBEREITUNG
99 # ============================
100
101 plt.style.use('default')
102 fig, (ax1, ax2) = plt.subplots(2, 1, figsize=(14, 11))
103 colors = plt.cm.tab10(np.linspace(0, 1, len(objects)))
104 debug_stats = []
105
106 # ============================
107 #  SIMULATION SCHLEIFE
108 # ============================
109
110 for idx, (name, M) in enumerate(objects):
111     print("\nSimuliere {} (M = {:.2e} M_sun)".format(name,
    M))
112     start_time = time.time()
113
114     deltas_ART = []
115     deltas_MOD = []
116     captured_ART = []
117     captured_MOD = []
118     phi_end_ART = []
119     phi_end_MOD = []
120
121     for i, b in enumerate(b_vals):
122         umax = solve_umax(b, rs)
123         if umax is None or umax <= 0:
124             deltas_ART.append(np.nan)
125             deltas_MOD.append(np.nan)
126             captured_ART.append(True)
127             captured_MOD.append(True)
128             phi_end_ART.append(np.nan)
```

```
            phi_end_MOD.append(np.nan)
            continue

        # --- ART ---
        captured_flag_art = True
        delta0_as_art = np.nan
        phi_end_val_art = np.nan
        try:
            sol_art = solve_ivp(
                geodesic_eq, [0, 10000], [umax, 0],
                args=(rs, 0.0, 0.0, 0.0),
                events=[event_u0, event_u_small,
    event_u_large],
                method='DOP853',
                rtol=1e-10, atol=1e-12,
                max_step=0.1
            )

            if len(sol_art.t_events[2]) > 0:  # u_large ->
    captured
                captured_flag_art = True
            elif len(sol_art.t_events[0]) > 0 or
    len(sol_art.t_events[1]) > 0:  # escaped
                phi_end_val_art = sol_art.t_events[0][0] if
    len(sol_art.t_events[0]) > 0 else sol_art.t_events[1][0]
                delta0_rad = 2 * (phi_end_val_art - np.pi/2)
                delta0_as_art = delta0_rad * arcsec_per_rad
                captured_flag_art = False
            else:
                if sol_art.y[0][-1] < 1e-5:
                    phi_end_val_art = sol_art.t[-1]
                    delta0_rad = 2 * (phi_end_val_art -
    np.pi/2)
                    delta0_as_art = delta0_rad *
    arcsec_per_rad
                    captured_flag_art = False
                else:
                    captured_flag_art = True

            phi_end_ART.append(phi_end_val_art)
        except Exception as e:
            print("  ART-Integration fehlgeschlagen bei
    b={:.3f}: {}".format(b, e))
            captured_flag_art = True
```

```
            phi_end_ART.append(np.nan)

        deltas_ART.append(delta0_as_art)
        captured_ART.append(captured_flag_art)

        # --- Modifiziert ---
        captured_flag_mod = True
        delta0_as_mod = np.nan
        phi_end_val_mod = np.nan
        try:
            sol_mod = solve_ivp(
                geodesic_eq, [0, 10000], [umax, 0],
                args=(rs, Qconst, Qlinear, Qquad),
                events=[event_u0, event_u_small,
    event_u_large],
                method='DOP853',
                rtol=1e-10, atol=1e-12,
                max_step=0.1
            )

            if len(sol_mod.t_events[2]) > 0:
                captured_flag_mod = True
            elif len(sol_mod.t_events[0]) > 0 or
    len(sol_mod.t_events[1]) > 0:
                phi_end_val_mod = sol_mod.t_events[0][0] if
    len(sol_mod.t_events[0]) > 0 else sol_mod.t_events[1][0]
                delta0_rad = 2 * (phi_end_val_mod - np.pi/2)
                delta0_as_mod = delta0_rad * arcsec_per_rad
                captured_flag_mod = False
            else:
                if sol_mod.y[0][-1] < 1e-5:
                    phi_end_val_mod = sol_mod.t[-1]
                    delta0_rad = 2 * (phi_end_val_mod -
    np.pi/2)
                    delta0_as_mod = delta0_rad *
    arcsec_per_rad
                    captured_flag_mod = False
                else:
                    captured_flag_mod = True

            phi_end_MOD.append(phi_end_val_mod)
        except Exception as e:
            print("  MOD-Integration fehlgeschlagen bei
    b={:.3f}: {}".format(b, e))
```

```
204             captured_flag_mod = True
205             phi_end_MOD.append(np.nan)
206
207         deltas_MOD.append(delta0_as_mod)
208         captured_MOD.append(captured_flag_mod)
209
210     # Arrays
211     deltas_ART = np.array(deltas_ART)
212     deltas_MOD = np.array(deltas_MOD)
213     captured_ART = np.array(captured_ART)
214     captured_MOD = np.array(captured_MOD)
215     phi_end_ART = np.array(phi_end_ART)
216     phi_end_MOD = np.array(phi_end_MOD)
217
218     # CSV Export
219     csv_filename = os.path.join(OUTPUT_DIR,
    "data_{}_final.csv".format(name))
220     with open(csv_filename, 'w', newline='',
    encoding='utf-8') as csvfile:
221         writer = csv.writer(csvfile)
222         writer.writerow([
223             "b_rs", "b_sim", "delta_ART_as", "delta_MOD_as",
224             "captured_ART", "captured_MOD", "phi_end_ART",
    "phi_end_MOD"
225         ])
226         for i in range(len(b_rs_vals)):
227             writer.writerow([
228                 b_rs_vals[i], b_vals[i],
229                 deltas_ART[i] if np.isfinite(deltas_ART[i])
    else "",
230                 deltas_MOD[i] if np.isfinite(deltas_MOD[i])
    else "",
231                 captured_ART[i], captured_MOD[i],
232                 phi_end_ART[i] if
    np.isfinite(phi_end_ART[i]) else "",
233                 phi_end_MOD[i] if
    np.isfinite(phi_end_MOD[i]) else ""
234             ])
235     print("CSV gespeichert: {}".format(csv_filename))
236
237     # Plot
238     mask_art = np.isfinite(deltas_ART)
239     mask_mod = np.isfinite(deltas_MOD)
240     if np.any(mask_art):
```

```
241         ax1.plot(b_rs_vals[mask_art], deltas_ART[mask_art],
242                  color=colors[idx], linestyle='-',
    linewidth=2.5,
243                  label='{} (ART)'.format(name))
244     if np.any(mask_mod):
245         ax1.plot(b_rs_vals[mask_mod], deltas_MOD[mask_mod],
246                  color=colors[idx], linestyle='--',
    linewidth=2.0,
247                  label='{} (Qlin={:.0e})'.format(name,
    Qlinear))
248 
249     with np.errstate(divide='ignore', invalid='ignore'):
250         rel_diff = (deltas_MOD - deltas_ART) / deltas_ART
251     mask_rel = np.isfinite(rel_diff) & (deltas_ART > 0)
252     if np.any(mask_rel):
253         ax2.plot(b_rs_vals[mask_rel], rel_diff[mask_rel] *
    100,
254                  color=colors[idx], linestyle='-',
    linewidth=2,
255                  label=name)
256 
257     # Debug-Statistik
258     runtime = time.time() - start_time
259     valid_ART = np.sum(mask_art)
260     valid_MOD = np.sum(mask_mod)
261     captured_count_ART = np.sum(captured_ART)
262     captured_count_MOD = np.sum(captured_MOD)
263     mean_rel_diff = np.nanmean(rel_diff[mask_rel]) * 100 if
    np.any(mask_rel) else np.nan
264     mean_delta_ART = np.nanmean(deltas_ART[mask_art]) if
    valid_ART > 0 else np.nan
265     mean_phi_ART =
    np.nanmean(phi_end_ART[np.isfinite(phi_end_ART)]) if
    np.any(np.isfinite(phi_end_ART)) else np.nan
266 
267     debug_stats.append({
268         "name": name,
269         "runtime_sec": runtime,
270         "valid_ART": valid_ART,
271         "valid_MOD": valid_MOD,
272         "captured_ART": captured_count_ART,
273         "captured_MOD": captured_count_MOD,
274         "mean_rel_diff_percent": mean_rel_diff,
275         "mean_delta_ART_as": mean_delta_ART,
```

```
        "mean_phi_ART": mean_phi_ART
    })

# === THEORIE & PHOTONENKREIS ===
ax1.plot(b_rs_vals, delta0_art_theory_as, 'k:', linewidth=3,
         label='ART Theorie: delta = 4 rs / b', zorder=100)
ax1.axvline(b_crit_rs, color='red', linestyle='-.',
            label='Photonenkreis (b/rs =
{:.3f})'.format(b_crit_rs))
ax2.axvline(b_crit_rs, color='red', linestyle='-.',
            linewidth=2)

# === PLOT LAYOUT ===
ax1.set_xlabel('Normierter Stossparameter b / r_s
(dimensionslos)', fontsize=13)
ax1.set_ylabel('Ablenkwinkel delta_0 (Bogensekunden)',
            fontsize=13)
ax1.set_title('Lichtablenkung in ART und modifizierter
Gravitation', fontsize=15, fontweight='bold')
ax1.set_yscale('log')
ax1.set_xscale('log')
ax1.grid(True, alpha=0.3)
ax1.legend(fontsize=8, loc='upper right')
ax1.set_ylim(50000, 500000)

ax2.set_xlabel('Normierter Stossparameter b / r_s',
            fontsize=13)
ax2.set_ylabel('Relative Abweichung [%]', fontsize=13)
ax2.set_title('Relative Abweichung durch Q_linear =
{:.0e}'.format(Qlinear), fontsize=14)
ax2.set_xscale('log')
ax2.grid(True, alpha=0.3)
ax2.legend(fontsize=10, loc='upper right')
ax2.axhline(0, color='black', linewidth=0.8, linestyle='-')

plt.tight_layout()
plot_path = os.path.join(OUTPUT_DIR,
            "lichtablenkung_verschiedene_massen.png")
plt.savefig(plot_path, dpi=300, bbox_inches='tight')
plt.show()

# ============================
#  DEBUG AUSGABE
```

```
# ============================

print("\n" + "="*80)
print("DEBUG-STATISTIK & AUSWERTUNG")
print("="*80)

for stat in debug_stats:
    print("\nObjekt: {}".format(stat['name']))
    print("   Laufzeit:           {:.2f}
    s".format(stat['runtime_sec']))
    print("   Gueltige ART-Werte: {} /
    {}".format(stat['valid_ART'], num_b))
    print("   Gueltige MOD-Werte: {} /
    {}".format(stat['valid_MOD'], num_b))
    print("   Eingefangen (ART):
    {}".format(stat['captured_ART']))
    print("   Eingefangen (MOD):
    {}".format(stat['captured_MOD']))
    if not np.isnan(stat['mean_rel_diff_percent']):
        print("   Mittlere Abweichung: {:+.4f}
    %".format(stat['mean_rel_diff_percent']))
    if not np.isnan(stat['mean_delta_ART_as']):
        print("   Mittlerer delta_0 (ART): {:.1f}
    Bogensek.".format(stat['mean_delta_ART_as']))
    if not np.isnan(stat['mean_phi_ART']):
        print("   Mittleres phi_end:   {:.1f}
    rad".format(stat['mean_phi_ART']))

print("\nPhotonenkreis bei b/rs = {:.6f} — rote Linie im
    Plot".format(b_crit_rs))
print("Alle Daten exportiert nach: {}".format(OUTPUT_DIR))
print("Plot gespeichert als: {}".format(plot_path))

print("\nSIMULATION ABGESCHLOSSEN.")
print("Ergebnisse:")
print("   - Universelle ART-Kurve fuer alle Massen")
print("   - Systematische Abweichung durch Q_linear
    sichtbar")
print("   - Nur sehr wenige Bahnen eingefangen (nahe am
    Photonenkreis)")
```

Listing A.7: Visualisierung Lichtablenkung verschiedener Massen

A.8 Geometrische Galaxienrotation, (Kap. 20)

```python
# galaxienrotation_geometrisch.py
import numpy as np
import matplotlib.pyplot as plt

# ===== PHYSIKALISCHE PARAMETER =====
G = 6.67430e-11           # Nur für Konsistenz – wird nicht verwendet!
c = 3.0e8                 # m/s
v_target = 200e3          # Ziel: 200 km/s = 200.000 m/s
K = 2 * v_target**2 / c**2  # K = 8.889e-7 → führt zu ∞v = 200 km/s
r0_kpc = 1.0              # Referenzradius – kann variiert werden, ändert nur Offset
kpc_to_m = 3.086e19       # 1 kpc in Meter

# ===== METRIKFUNKTION – NUR LOG-TERM, KEIN NEWTON =====
def F(r_kpc):
    if r_kpc <= 0:
        return 1.0
    r_m = r_kpc * kpc_to_m
    r0_m = r0_kpc * kpc_to_m

    #  KEIN NEWTON-Term – nur geometrischer log-Term
    term = K * np.log(r_m / r0_m)

    return 1.0 + term  # Kein -2GM/c²r !

# ===== ROTATIONSGESCHWINDIGKEIT =====
def rotation_velocity(r_kpc):
    dr = 0.1  # kpc
    F_plus = F(r_kpc + dr/2)
    F_minus = F(r_kpc - dr/2)
    dF_dr = (F_plus - F_minus) / dr  # [1/kpc]
    dF_dr_SI = dF_dr / kpc_to_m

    r_m = r_kpc * kpc_to_m
    v_squared = (c**2 * r_m / 2.0) * dF_dr_SI

    if v_squared < 0:
        return 0.0
```

```
    return np.sqrt(v_squared) / 1000  # km/s

# ===== SIMULATION =====
print("□ Galaxy rotation – PURE LOGARITHMIC METRIC (NO NEWTON – PERFECTLY FLAT)")
print(f"Target: {v_target/1000:.0f} km/s → K = {K:.3e}, r0 = {r0_kpc} kpc")
print(f"Note: Baryonic mass term is DISABLED – pure geometric dark □□matter")

r_min, r_max, steps = 1.0, 50.0, 100
r_values = np.linspace(r_min, r_max, steps)
v_values = np.array([rotation_velocity(r) for r in r_values])

# ===== PLOT =====
plt.figure(figsize=(12, 7))
plt.plot(r_values, v_values, 'b-o', linewidth=2.5, markersize=3, label='Reines log-Potential (kein Newton)')
asymptote = np.mean(v_values[-10:])
plt.axhline(y=asymptote, color='r', linestyle='--', label=f'Asymptotisch: {asymptote:.3f} km/s')
plt.axhline(y=v_target/1000, color='green', linestyle=':', label=f'Ziel: {v_target/1000:.0f} km/s')
plt.xlabel('Radius $r$ [kpc]', fontsize=14)
plt.ylabel('Rotationsgeschwindigkeit $v(r)$ [km/s]', fontsize=14)
plt.title('Galaxienrotation – Perfekt flache Kurve durch reines log-Potential (keine Dunkle Materie)', fontsize=14)
plt.grid(True, alpha=0.4)
plt.legend(fontsize=13)
plt.ylim(190, 210)  # Zoom auf den flachen Bereich
plt.tight_layout()
plt.savefig("galaxienrotation_log_pure.png", dpi=200)
plt.show()
plt.close()

# ===== ERGEBNISSE =====
v_initial = v_values[0] if len(v_values) > 0 else 0.0
v_at_5kpc = v_values[np.argmin(np.abs(r_values - 5.0))] if len(v_values) > 10 else 0.0
v_final = asymptote
print(f"\□□□n PERFEKT FLACH – WIRKLICH, WIRKLICH, WIRKLICH!")
print(f"Velocity at 1.0 kpc: {v_initial:.3f} km/s")
print(f"Velocity at 5.0 kpc: {v_at_5kpc:.3f} km/s")
```

```
print(f"Final (asymptotic) velocity: {v_final:.3f} km/s")
print(f"Flachheit der Kurve (std der letzten 10 Punkte):
    {np.std(v_values[-10:]):.5f} km/s")
```

Listing A.8: Visualisierung Geometrische Galaxienrotation

A.9 Flache Galaxienrotation (Animation), (Abschn. 21.1)

```
# flache_galaxienrotation_animation.py
# Verbesserte Animation: Flache Rotationskurve durch
    geometrischen Effekt
# MIT DYNAMISCHEM RECHTEN PLOT, der die aktuelle
    Sternposition live anzeigt!

import numpy as np
import matplotlib.pyplot as plt
import matplotlib.animation as animation
from matplotlib.patches import Circle

# ===== PHYSIKALISCHE PARAMETER =====
c = 3.0e8  # m/s
v_target = 200e3  # Ziel: 200 km/s = 200.000 m/s
K = 2 * v_target**2 / c**2  # K = 8.889e-7 → führt zu ∞v_ =
    200 km/s
r0_kpc = 1.0  # Referenzradius
kpc_to_m = 3.086e19  # 1 kpc in Meter

# ===== METRIKFUNKTIONEN =====
def F_modified(r_kpc):
    """Modifizierte Metrik mit logarithmischem Term (Kapitel
    19)."""
    if r_kpc <= 0:
        return 1.0
    r_m = r_kpc * kpc_to_m
    r0_m = r0_kpc * kpc_to_m
    term = K * np.log(r_m / r0_m)
    return 1.0 + term

# ===== ROTATIONSGESCHWINDIGKEITEN =====
def rotation_velocity_modified(r_kpc):
```

```
    """Rotationsgeschwindigkeit aus der modifizierten Metrik (flach)."""
    dr = 0.1  # kpc
    F_plus = F_modified(r_kpc + dr/2)
    F_minus = F_modified(r_kpc - dr/2)
    dF_dr = (F_plus - F_minus) / dr  # [1/kpc]
    dF_dr_SI = dF_dr / kpc_to_m

    r_m = r_kpc * kpc_to_m
    v_squared = (c**2 * r_m / 2.0) * dF_dr_SI

    if v_squared < 0:
        return 0.0
    return np.sqrt(v_squared) / 1000  # km/s

# ===== SIMULATION =====
print("Erstelle dynamische Animation: Flache Galaxienrotationskurve durch geometrischen Effekt")
print(f"Zielgeschwindigkeit: {v_target/1000:.0f} km/s")

# Simulationsbereich für das Diagramm
r_min, r_max, steps = 0.5, 25.0, 100
r_values = np.linspace(r_min, r_max, steps)
v_modified = np.array([rotation_velocity_modified(r) for r in r_values])

# Erstelle eine feste Menge von Sternen für die Animation
num_stars = 25
star_radii = np.linspace(3, 20, num_stars)  # Radien der Sterne in kpc
star_angles = np.random.uniform(0, 2*np.pi, num_stars)  # Startwinkel

# ===== ANIMATION SETUP =====
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(16, 8))
fig.suptitle('Flache Galaxienrotationskurve durch geometrischen Effekt\n"Keine Dunkle Materie nötig"',
             fontsize=14, fontweight='bold')

# Separate Zeile für die Visualisierungsinformation
fig.text(0.27, 0.83, 'Visualisierung: Klaus H. Dieckmann, 2025',
         fontsize=12, style='italic', ha='center')

```

```
# --- Linkes Diagramm: Schematische Galaxie ---
ax1.set_xlim(-25, 25)
ax1.set_ylim(-25, 25)
ax1.set_aspect('equal')
ax1.set_xlabel('X [kpc]', fontsize=12)
ax1.set_ylabel('Y [kpc]', fontsize=12)
ax1.grid(True, alpha=0.2)

# Zeichne den zentralen Bulge
central_bulge = Circle((0, 0), 2.0, color='gold', alpha=0.8,
    label='Galaktischer Bulge')
ax1.add_patch(central_bulge)

# Erstelle Stern-Objekte
star_markers = []
for i in range(num_stars):
    star, = ax1.plot([], [], 'o', markersize=8, alpha=0.7)
    star_markers.append(star)

# --- Rechtes Diagramm: Rotationskurven (JETZT DYNAMISCH!)
    ---
ax2.plot(r_values, v_modified, 'b-', linewidth=3,
    label='Reguläre Kern-Metrik: flache Kurve')
ax2.axhline(y=v_target/1000, color='green', linestyle=':',
    linewidth=2, label=f'Ziel: {v_target/1000:.0f} km/s')

# Füge einen Punkt hinzu, der die aktuelle Position des
    ersten Sterns im Diagramm zeigt
live_point, = ax2.plot([], [], 'ro', markersize=12,
    label='Aktuelle Position von Stern #1')

ax2.set_xlabel('Abstand vom Zentrum $r$ [kpc]', fontsize=12)
ax2.set_ylabel('Rotationsgeschwindigkeit $v(r)$ [km/s]',
    fontsize=12)
ax2.set_title('Rotationsgeschwindigkeitsprofil $v(r)$',
    fontsize=12)
ax2.grid(True, alpha=0.3)
ax2.legend(fontsize=10, loc='lower right')
ax2.set_ylim(180, 220)  # Fokus auf den flachen Bereich

# ===== ANIMATIONSFUNKTION =====
def animate(frame):
    # Aktualisiere die Position jedes Sterns
    for i, radius in enumerate(star_radii):
```

```
        # Winkel aktualisieren
        v_current = rotation_velocity_modified(radius) * 1000  # zurück in m/s
        angular_velocity = v_current / (radius * kpc_to_m)  # rad/s
        star_angles[i] += angular_velocity * 1e14

        x = radius * np.cos(star_angles[i])
        y = radius * np.sin(star_angles[i])

        # Geschwindigkeit bestimmt die Farbe
        norm_v = (v_current / 1000) / 220  # Normalisierung für Farbpalette
        color = plt.cm.coolwarm(norm_v)

        star_markers[i].set_data([x], [y])
        star_markers[i].set_color(color)
        star_markers[i].set_markersize(6 + 4 * norm_v)  # Größe nach Geschwindigkeit

    # ===== DYNAMISCHE AKTUALISIERUNG DES RECHTEN PLOTS =====
    # Zeige die Position und Geschwindigkeit des ersten Sterns im Diagramm
    current_radius = star_radii[0]
    current_velocity = rotation_velocity_modified(current_radius)

    live_point.set_data([current_radius], [current_velocity])

    return star_markers + [live_point]

# ===== ANIMATION ERSTELLEN UND SPEICHERN =====
print("Erstelle dynamisches GIF...")

ani = animation.FuncAnimation(
    fig,
    animate,
    frames=200,
    interval=50,
    blit=False,
    repeat=True
)

# GIF speichern
```

```
ani.save('flache_galaxienrotation_animation.gif',
    writer='pillow', fps=20, dpi=150)

plt.tight_layout()
plt.show()

print("✅ Dynamische Animation
    'flache_galaxienrotation_animation_dynamisch.gif'
    erfolgreich erstellt!")
print("Der rote Punkt im rechten Diagramm zeigt live die
    Position und Geschwindigkeit")
print("des ersten Sterns an und veranschaulicht so direkt
    den Zusammenhang")
print("zwischen Radius und der flachen
    Rotationsgeschwindigkeit.")
```

Listing A.9: Visualisierung Flache Galaxienrotation (Animation)

A.10 Geodäten im Planckregime, (Abschn. 23.4.2)

```
# geodaeten_im_planckregime.py
import numpy as np
from scipy.integrate import solve_ivp
from scipy.optimize import fsolve, root
import matplotlib.pyplot as plt

# Physikalische Konstanten
G = 6.67430e-11  # m^3 kg^-1 s^-2
c = 299792458    # m/s (genauer Wert)
lp = 1.616255e-35  # Planck-Länge, m
M_planck = np.sqrt(c * lp / G)  # Planck-Masse

print("=" * 60)
print("PHYSIKALISCHE KONSTANTEN")
print("=" * 60)
print(f"Planck-Länge: {lp:.3e} m")
print(f"Lichtgeschwindigkeit: {c:.3e} m/s")
print(f"Gravitationskonstante: {G:.3e} m³/kg/s²")
print(f"Planck-Masse: {M_planck:.3e} kg")

# Schwarzschildradius für Planck-Masse
r_s = (2 * G * M_planck) / c**2
print(f"Schwarzschildradius für Planck-Masse: {r_s:.3e} m")
```

```
print(f"Schwarzschildradius in Planck-Längen: {r_s/lp:.3f}")

# Wir verwenden die Planck-Masse für die Simulation
M = M_planck
print(f"\nVerwendete Masse: {M:.3e} kg (Planck-Masse)")

# Modifizierte Metrikfunktion für die Planck-Skala
def F(r, alpha, beta):
    """
    r: dimensionsloser Radius (in Einheiten von lp)
    alpha, beta: Korrekturparameter für Quanteneffekte
    """
    # Schutz gegen zu kleine und zu große Radien
    r = np.clip(r, 0.1, 1000)

    # Schwarzschild-Term: 1 - r_s/r
    # r_s = 2 für Planck-Masse in unseren Einheiten
    schwarzschild_term = 1 - 2/r

    # Quantenkorrekturen: repulsive Terme, die bei kleinen Radien dominieren
    # Wir verwenden eine Regularisierung, die bei r=0 endliche Werte liefert
    quantum_repulsion = alpha * np.exp(-beta * (r - 1)**2) / (1 + r**4)

    return schwarzschild_term + quantum_repulsion

# Ableitung der Metrikfunktion (numerisch stabiler)
def dF_dr(r, alpha, beta):
    """
    Ableitung der dimensionslosen Metrikfunktion
    """
    # Schutz gegen zu kleine und zu große Radien
    r = np.clip(r, 0.1, 1000)

    # Numerische Ableitung für bessere Stabilität
    h = 1e-6
    return (F(r + h, alpha, beta) - F(r - h, alpha, beta)) / (2 * h)

# Geodäten-Gleichungen
def geodaten(t, y, alpha, beta):
    r, dr_dt = y
```

```
64
65      # Schutz gegen zu kleine Radien
66      if r < 0.5:
67          return [0, 0]
68
69      # Beschleunigung berechnen: d²r/dt² = -1/2 * dF/dr
70      d2r_dt2 = -0.5 * dF_dr(r, alpha, beta)
71
72      return [dr_dt, d2r_dt2]
73
74  # Simulationsparameter
75  alpha, beta = 5.0, 5.0  # Stärke und Reichweite der
        Quantenabstoßung
76  r0 = 4.0    # Startradius (4 Planck-Längen)
77  v0 = 0.0    # Anfangsgeschwindigkeit
78  t_span = (0, 100)  # Dimensionslose Zeit
79
80  print("\n" + "=" * 60)
81  print("SIMULATIONSPARAMETER")
82  print("=" * 60)
83  print(f"Korrekturparameter alpha: {alpha}")
84  print(f"Korrekturparameter beta: {beta}")
85  print(f"Startradius: {r0} lp")
86  print(f"Anfangsgeschwindigkeit: {v0}")
87
88  # Simulation durchführen
89  sol = solve_ivp(geodaten, t_span, [r0, v0], args=(alpha,
        beta),
90                  method='DOP853', rtol=1e-8, atol=1e-10,
91                  dense_output=True, max_step=0.1)
92
93  # Umrechnung in physikalische Einheiten
94  time_scale = lp / c  # Charakteristische Zeit
95  t_phys = sol.t * time_scale
96  r_phys = sol.y[0] * lp
97
98  print("\n" + "=" * 60)
99  print("ERGEBNISSE")
100 print("=" * 60)
101 print(f"Anzahl der Zeitschritte: {len(sol.t)}")
102 print(f"Simulationsdauer: {t_phys[-1]:.3e} s")
103 print(f"Endradius: {sol.y[0][-1]:.3f} lp")
104 print(f"Endgeschwindigkeit: {sol.y[1][-1]:.3e}")
105
```

```
# Berechnung der physikalischen Beschleunigung
accelerations = []
for i in range(len(sol.t)):
    r, dr_dt = sol.y[0][i], sol.y[1][i]
    if r > 0.5:
        # Dimensionslose Beschleunigung in physikalische umrechnen
        accel_dimless = -0.5 * dF_dr(r, alpha, beta)
        accel_phys = accel_dimless * (lp / time_scale**2)
        accelerations.append(accel_phys)
    else:
        accelerations.append(0)

print(f"Maximale Beschleunigung: {np.max(np.abs(accelerations)):.3e} m/s²")

# Metrik am Startpunkt analysieren
F_start = F(r0, alpha, beta)
dF_start = dF_dr(r0, alpha, beta)
print(f"Metrik F(r_start): {F_start:.3e}")
print(f"Ableitung dF/dr(r_start): {dF_start:.3e}")

# Visualisierung
plt.figure(figsize=(15, 10))

# Radiusentwicklung
plt.subplot(2, 2, 1)
plt.plot(t_phys, sol.y[0], 'b-', linewidth=2)
plt.axhline(y=2, color='r', linestyle='--', label='Ereignishorizont (2 lp)')
plt.axhline(y=1, color='g', linestyle='--', label='Planck-Länge (1 lp)')
plt.xlabel('Zeit [s]')
plt.ylabel('Radius [lp]')
plt.title('Entwicklung des Radius')
plt.legend()
plt.grid(True)

# Geschwindigkeitsentwicklung
plt.subplot(2, 2, 2)
plt.plot(t_phys, sol.y[1], 'g-', linewidth=2)
plt.xlabel('Zeit [s]')
plt.ylabel('Geschwindigkeit [dimensionslos]')
plt.title('Radialgeschwindigkeit')
```

```
plt.grid(True)

# Beschleunigung
plt.subplot(2, 2, 3)
plt.plot(t_phys, accelerations, 'r-', linewidth=2)
plt.xlabel('Zeit [s]')
plt.ylabel('Beschleunigung [m/s²]')
plt.title('Radialbeschleunigung')
plt.yscale('log')
plt.grid(True)

# Metrikfunktion
plt.subplot(2, 2, 4)
F_values = [F(r, alpha, beta) for r in sol.y[0]]
plt.plot(t_phys, F_values, 'm-', linewidth=2)
plt.xlabel('Zeit [s]')
plt.ylabel('F(r)')
plt.title('Metrikfunktion')
plt.grid(True)

plt.tight_layout()
plt.suptitle('Geodäten im Planck-Regime mit
    Quantenkorrekturen', y=1.02, fontsize=16)
plt.show()

# Detaillierte Analyse
print("\n" + "=" * 60)
print("DETAILANALYSE")
print("=" * 60)
print("Zeitliche Entwicklung:")
indices = np.linspace(0, len(sol.t)-1, min(6, len(sol.t)),
    dtype=int)
for i in indices:
    r_val = sol.y[0][i]
    v_val = sol.y[1][i]
    F_val = F(r_val, alpha, beta)
    print(f"t={t_phys[i]:.2e}s: r={r_val:.3f}lp,
    v={v_val:.3e}, F(r)={F_val:.3e}")

# Effektives Potential analysieren
r_range = np.linspace(0.5, 10, 1000)
F_range = [F(r, alpha, beta) for r in r_range]
dF_range = [dF_dr(r, alpha, beta) for r in r_range]

```

```
plt.figure(figsize=(12, 5))

plt.subplot(1, 2, 1)
plt.plot(r_range, F_range, 'b-', linewidth=2)
plt.axvline(x=2, color='r', linestyle='--',
    label='Ereignishorizont')
plt.axvline(x=1, color='g', linestyle='--',
    label='Planck-Länge')
plt.xlabel('Radius [lp]')
plt.ylabel('F(r)')
plt.title('Metrikfunktion')
plt.grid(True)
plt.legend()

plt.subplot(1, 2, 2)
plt.plot(r_range, dF_range, 'g-', linewidth=2)
plt.axvline(x=2, color='r', linestyle='--',
    label='Ereignishorizont')
plt.axvline(x=1, color='g', linestyle='--',
    label='Planck-Länge')
plt.xlabel('Radius [lp]')
plt.ylabel('dF/dr')
plt.title('Ableitung der Metrikfunktion')
plt.grid(True)
plt.legend()

plt.tight_layout()
plt.show()

# Finde Gleichgewichtspunkte (wo dF/dr = 0)
def find_equilibrium(alpha, beta):
    """Finde Gleichgewichtspunkte wo dF/dr = 0"""
    def equation(r):
        return dF_dr(r, alpha, beta)

    # Startwerte für die Suche
    r_guess = [1.5, 3.0, 5.0]
    equilibria = []

    for guess in r_guess:
        try:
            # Verwende root statt fsolve für bessere
   Kontrolle
```

```
            result = root(equation, guess, method='hybr',
    options={'xtol': 1e-10})
            if result.success and 0.5 < result.x[0] < 10:
                r_eq = result.x[0]
                if abs(equation(r_eq)) < 1e-5:
                    equilibria.append(r_eq)
        except:
            continue

    return np.unique(np.round(equilibria, 3))

equilibrium_points = find_equilibrium(alpha, beta)
print(f"\nGleichgewichtspunkte (dF/dr = 0):
    {equilibrium_points} lp")

# Stabilitätsanalyse an den Gleichgewichtspunkten
for r_eq in equilibrium_points:
    # Numerische Ableitung von dF/dr an diesen Punkten
    h = 1e-5
    d2F_dr2 = (dF_dr(r_eq + h, alpha, beta) - dF_dr(r_eq -
    h, alpha, beta)) / (2 * h)

    if d2F_dr2 > 0:
        stability = "stabiles Gleichgewicht"
    else:
        stability = "instabiles Gleichgewicht"

    print(f"Punkt r = {r_eq:.3f} lp: {stability} (d²F/dr² =
    {d2F_dr2:.3e})")
```

Listing A.10: Visualisierung Geodäten im Planckregime

A.11 Fall in ein schwarzes Loch, (Abschn. 22.2)

```
# schwarzes_loch_fall_gif_animation.py
# Animation: Fall in ein reguläres schwarzes Loch (mit h > 0)
# Basierend auf der modifizierten Schwarzschild-Metrik
# F(r) = 1 - 2/(r - h)  [in natürlichen Einheiten: G=M=c=1]

import numpy as np
import matplotlib.pyplot as plt
```

```
import matplotlib.animation as animation
from scipy.integrate import solve_ivp

# --- 1. Definition der Metrikfunktion F(r)  ---
def F(r, h=1.0):
    """Reguläre Kern-Metrikfunktion: F(r) = 1 - 2/(r - h)"""
    if r <= h:
        return 1e6  # Sehr großer Wert, um starke Abstoßung bei r <= h zu simulieren
    return 1 - 2 / (r - h)

def dF_dr(r, h=1.0):
    """Analytische Ableitung von F(r): dF/dr = 2 / (r - h)^2"""
    if r <= h:
        return -1e6  # Sehr große negative Ableitung -> starke Abstoßung
    return 2 / ((r - h) ** 2)

# --- 2. Geodätengleichung für radiale Bewegung  ---
def geodaten(t, y, h):
    """
    Gibt die Ableitungen [dr/dt, d²r/dt²] zurück.
    Für zeitartige Geodäten gilt: d²r/dt² = - (1/2) * dF/dr
    """
    r, dr_dt = y

    # Sicherheitscheck: Verhindere, dass r zu klein wird
    if r <= h + 1e-3:
        return [0, 0]  # Stoppe die Bewegung nahe dem Kernradius

    # Radiale Beschleunigung berechnen
    d2r_dt2 = -0.5 * dF_dr(r, h)  # Physikalische Gleichung

    return [dr_dt, d2r_dt2]

# --- 3. Simulationsparameter ---
h_val = 1.0           # Kernradius (in natürlichen Einheiten)
r0 = 5.0              # Startposition (r > h)
v0 = -0.8             # Anfangsgeschwindigkeit (negativ = Richtung Zentrum)
t_span = (0, 25)      # Simulationszeit

```

```
# --- 4. Numerische Integration der Geodätengleichung ---
sol = solve_ivp(
    geodaten,
    t_span,
    [r0, v0],
    args=(h_val,),
    method='DOP853',
    rtol=1e-8,
    atol=1e-10,
    dense_output=True
)

# --- 5. Vorbereitung für die Animation ---
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(16, 7))

# --- Linkes Diagramm: Raumzeit-Darstellung ---
ax1.set_xlim(0, 6)
ax1.set_ylim(-1.5, 1.5)
ax1.set_aspect('equal')
ax1.set_xlabel('x', fontsize=12)
ax1.set_ylabel('y', fontsize=12)
ax1.set_title('Testteilchen in Reguläre Kern-Metrik: $F(r) =
    1 - \\frac{2}{r - h}$', fontsize=12)
ax1.grid(True, alpha=0.3)

# Zeichne das zentrale Objekt (Ereignishorizont bei r=2) und
    den Kern (bei r=h)
central_body = plt.Circle((0, 0), 2.0, color='black',
    alpha=0.7, label='Ereignishorizont (r=2)')
core = plt.Circle((0, 0), h_val, color='red', alpha=0.9,
    label=f'Kernradius h={h_val}')
ax1.add_patch(central_body)
ax1.add_patch(core)
particle, = ax1.plot([], [], 'bo', markersize=10,
    label='Testteilchen')
ax1.legend(loc='upper right', fontsize=10)

# --- Rechtes Diagramm: Plots von F(r) und Beschleunigung
    a_r ---
r_plot = np.linspace(h_val + 0.1, 6, 500)
F_plot = [F(r, h_val) for r in r_plot]
a_plot = [-0.5 * dF_dr(r, h_val) for r in r_plot]  # a_r =
    d²r/dt²

```

```
ax2_twin = ax2.twinx()
line_F, = ax2.plot(r_plot, F_plot, 'b-', label='$F(r)$ Reguläre Kern-Metrik', linewidth=2)
line_a, = ax2_twin.plot(r_plot, a_plot, 'r-', label='Beschleunigung $a_r$', linewidth=2)
ax2.set_xlabel('Radius $r$', fontsize=12)
ax2.set_ylabel('$F(r)$', color='b', fontsize=12)
ax2_twin.set_ylabel('$a_r$', color='r', fontsize=12)
ax2.set_title('Reguläre Kern-Metrik und radiale Beschleunigung', fontsize=12)
ax2.grid(True, alpha=0.3)

# Vertikale Linie für aktuelle Position
current_pos_F, = ax2.plot([], [], 'k--', linewidth=2, label='Aktuelle Position')
current_pos_a, = ax2_twin.plot([], [], 'k--', linewidth=2)

# Legende nach unten verschieben
lines1, labels1 = ax2.get_legend_handles_labels()
lines2, labels2 = ax2_twin.get_legend_handles_labels()
ax2.legend(
    lines1 + lines2,
    labels1 + labels2,
    loc='upper center',
    bbox_to_anchor=(0.5, 0.2),   # Noch weiter runter
    fontsize=9,
    frameon=True,
    fancybox=True,
    shadow=True,
    ncol=1
)

# --- 6. Dynamischer Erklärtext (unten im Plot) ---
explanation_text = ax1.text(
    0.5, -0.2,  # Position unten in der Mitte
    '',
    transform=ax1.transAxes,
    fontsize=11,
    ha='center',
    va='top',
    bbox=dict(boxstyle="round,pad=0.3", facecolor="lightyellow", alpha=0.9),
    wrap=True
)
```

```
123
124 # --- 7. Statischer Hinweis oben im Plot  ---
125 author_text = ax1.text(
126     0.5, 1.2,  # Position oben in der Mitte
127     'Visualisierung: Klaus H. Dieckmann, 2025',
128     transform=ax1.transAxes,
129     fontsize=12,
130     ha='center',
131     va='bottom',
132     #weight='bold',
133     style='italic'
134 )
135
136 # --- 8. Animationsfunktion mit Phasen-Erklärung ---
137 def animate(i):
138     # Aktuelle Zeit und Position
139     t = sol.t[i]
140     r = sol.y[0][i]
141     v = sol.y[1][i]
142
143     # Update Teilchenposition (auf x-Achse für radiale
    Bewegung)
144     particle.set_data([r], [0])
145
146     # Update vertikale Linien in den Plots
147     current_pos_F.set_data([r, r], [min(F_plot),
    max(F_plot)])
148     current_pos_a.set_data([r, r], [min(a_plot),
    max(a_plot)])
149
150     # --- Dynamischer Erklärtext (Phasenlogik) ---
151     phase_text = ""
152     if r > 3.0:
153         phase_text = "Phase 1: Das Teilchen beginnt seinen
    freien Fall Richtung Zentrum.\nDie Gravitation zieht es
    an, die Beschleunigung ist negativ."
154     elif r > h_val + 0.5:
155         phase_text = "Phase 2: Das Teilchen nähert sich dem
    Kernradius h.\nDie Anziehungskraft wird stärker, die
    Geschwindigkeit nimmt zu."
156     elif r > h_val + 0.1:
157         phase_text = "Phase 3: KRITISCHER BEREICH – Das
    Teilchen erreicht den Kernradius h.\nEin starker,
    repulsiver Quanteneffekt (nach Dieckmann 2025) setzt
```

```
    ein.\nDie Beschleunigung wird heftig positiv (abstoßend)."
158     elif v > 0:  # Teilchen bewegt sich weg
159         phase_text = "Phase 4: ABPRALL – Der repulsive
    Effekt hat das Teilchen gestoppt und zurückgeworfen.\nEs
    entfernt sich nun wieder vom Zentrum. Die Singularität
    wurde vermieden!"
160     else:
161         phase_text = "Phase 3 (Fortsetzung): Maximale
    Kompression – Das Teilchen ist nahezu zum Stillstand
    gekommen.\nDer repulsive Druck überwindet gerade die
    Gravitation."
162
163     explanation_text.set_text(phase_text)
164
165     return particle, current_pos_F, current_pos_a,
    explanation_text
166
167 # --- 9. Animation erstellen und speichern ---
168 ani = animation.FuncAnimation(
169     fig,
170     animate,
171     frames=len(sol.t),
172     interval=300,  # Millisekunden pro Frame
173     blit=False,   # Wichtig, da der Text dynamisch ist und
    nicht geblittet werden kann
174     repeat=True
175 )
176
177 # GIF speichern
178 print("Speichere Animation als
    'schwarzes_loch_fall_animation.gif'...")
179 ani.save('schwarzes_loch_fall_animation.gif',
    writer='pillow', fps=3)
180
181 plt.tight_layout()
182 plt.subplots_adjust(bottom=0.2)  # Platz für den Erklärtext
    unten
183 plt.show()
184
185 print("□ Animation erfolgreich erstellt!")
186 print("Die Animation visualisiert den Mechanismus der
    Singularitätsvermeidung")
187 print("durch einen endlichen Kernradius h > 0, wie in
    Reguläre Kern-Metrik postuliert.")
```

Listing A.11: Visualisierung Fall in ein schwarzes Loch

A.12 Bahnenvergleich mit verschiedenen Metriken (Animation), (Abschn. 17.2)

```
# bahnen_vergleich_metriken_animation.py
# Animation: Vergleich von Bahnen in Newton, Schwarzschild-ART und regulärer Kern-Metrik
# Visualisiert den Fortschritt der Modelle: Newton → ART → Modifizierte Metrik

import numpy as np
import matplotlib.pyplot as plt
import matplotlib.animation as animation
from scipy.integrate import solve_ivp
from scipy.optimize import root

# ===== PHYSIKALISCHE PARAMETER (NATÜRLICHE EINHEITEN: G = c = M = 1) =====
rs = 2.0  # Schwarzschild-Radius
a_mercury = 3.92e7  # große Halbachse (in Einheiten von GM/c²)
e_mercury = 0.2056  # Exzentrizität

# Berechne Perihel und Aphel
r_min = a_mercury * (1 - e_mercury)
r_max = a_mercury * (1 + e_mercury)
u_min = 1 / r_max
u_max = 1 / r_min

print(f"Perihel (r_min): {r_min:.3f}")
print(f"Aphel (r_max): {r_max:.3f}")

# ===== BERECHNUNG VON ENERGIE UND DREHIMPULS FÜR SCHWARZSCHILD =====
def solve_EL_GR():
    def equations(x):
        L, E = x
        eq1 = E**2 - (1 - rs/r_min) * (1 + L**2 / r_min**2)
```

```
        eq2 = E**2 - (1 - rs/r_max) * (1 + L**2 / r_max**2)
        return [eq1, eq2]

    L_newton = np.sqrt(a_mercury * (1 - e_mercury**2))
    E_newton = np.sqrt(1 - rs/(2*a_mercury))
    sol = root(equations, [L_newton, E_newton], method='lm')
    if not sol.success:
        raise RuntimeError("Konvergenz bei E und L fehlgeschlagen!")
    return sol.x[0], sol.x[1]

L, E = solve_EL_GR()
print(f"Energie E: {E:.12f}")
print(f"Drehimpuls L: {L:.6f}")

# ===== BAHNGLEICHUNGEN FÜR DIE DREI MODELLE =====

# 1. NEWTONSCHES MODELL
def newton_orbit_equation(phi, y, L_sq):
    u, v = y
    dudphi = v
    dvdphi = -u + 1/L_sq  # Kein GR-Korrekturterm
    return [dudphi, dvdphi]

# 2. SCHWARZSCHILD-ART
def gr_orbit_equation(phi, y, L_sq):
    u, v = y
    dudphi = v
    dvdphi = -u + 1.5 * rs * u**2 + 1/L_sq  # ART: + 3/2 rs u²
    return [dudphi, dvdphi]

# 3. MODIFIZIERTE METRIK (REGULÄRE KERN-METRIK mit kleiner Korrektur)
# Wir fügen einen winzigen quadratischen Korrekturterm hinzu (z.B. b2 * u^2 mit b2 = 1e-10)
# Dies simuliert eine subtile Abweichung, wie sie in Kapitel 17.5 diskutiert wird.
def modified_orbit_equation(phi, y, L_sq, b2=1e-10):
    u, v = y
    dudphi = v
    # ART-Term + winziger zusätzlicher Term (z.B. aus Q(r) = b2 * r^2)
    dvdphi = -u + 1.5 * rs * u**2 + 1/L_sq + b2 * u**2
```

```
    return [dudphi, dvdphi]

# ===== NUMERISCHE INTEGRATION =====
def integrate_orbit(model_func, L_val, b2=0.0, num_orbits=3):
    L_sq = L_val**2
    u0, v0 = u_max, 0.0  # Start am Perihel
    phi_span = [0, 2 * np.pi * num_orbits]

    if b2 == 0.0:
        sol = solve_ivp(
            lambda phi, y: model_func(phi, y, L_sq),
            phi_span, [u0, v0],
            method='DOP853',
            rtol=1e-10, atol=1e-10,
            max_step=0.01,
            dense_output=True
        )
    else:
        sol = solve_ivp(
            lambda phi, y: model_func(phi, y, L_sq, b2),
            phi_span, [u0, v0],
            method='DOP853',
            rtol=1e-10, atol=1e-10,
            max_step=0.01,
            dense_output=True
        )

    if not sol.success:
        raise RuntimeError("Integration fehlgeschlagen!")
    return sol

# Integriere alle drei Bahnen
print("Integriere Newtonsche Bahn...")
sol_newton = integrate_orbit(newton_orbit_equation, L,
    num_orbits=3)

print("Integriere Schwarzschild-ART Bahn...")
sol_gr = integrate_orbit(gr_orbit_equation, L, num_orbits=3)

print("Integriere Modifizierte Metrik Bahn...")
sol_modified = integrate_orbit(modified_orbit_equation, L,
    b2=1e-10, num_orbits=3)

# ===== ANIMATION SETUP =====
```

```
fig, axes = plt.subplots(1, 3, figsize=(18, 6))
#fig.suptitle('Vergleich von Bahnen in verschiedenen Gravitationsmodellen\nVisualisierung: Klaus H. Dieckmann, 2025', fontsize=14, fontweight='bold')
fig.suptitle('Vergleich von Bahnen in verschiedenen Gravitationsmodellen',
             fontsize=14, fontweight='bold')

# Separate Zeile für die Visualisierungsinformation
fig.text(0.5, 0.91, 'Visualisierung: Klaus H. Dieckmann, 2025',
         fontsize=12, style='italic', ha='center')

titles = [
    '1. Newtonsche Gravitation\n(perfekte, geschlossene Ellipse)',
    '2. Schwarzschild-ART\n(relativistische Periheldrehung)',
    '3. Reguläre Kern-Metrik\n(subtile Abweichung durch Q(r))'
]

colors = ['green', 'blue', 'red']
labels = ['Newtonsche Bahn', 'Schwarzschild-ART', 'Modifizierte Metrik']

for i, ax in enumerate(axes):
    ax.set_xlim(-1.2 * r_max, 1.2 * r_max)
    ax.set_ylim(-1.2 * r_max, 1.2 * r_max)
    ax.set_aspect('equal')
    ax.set_xlabel('X [Einheiten von GM/c²]', fontsize=10)
    ax.set_ylabel('Y [Einheiten von GM/c²]', fontsize=10)
    ax.set_title(titles[i], fontsize=11)
    ax.grid(True, alpha=0.3)

# Erstelle leere Linienobjekte für die Bahnen
orbit_lines = []
particle_dots = []

for i, ax in enumerate(axes):
    line, = ax.plot([], [], '-', color=colors[i], linewidth=2, label=labels[i])
    dot, = ax.plot([], [], 'o', color=colors[i], markersize=8)
    orbit_lines.append(line)
```

```
    particle_dots.append(dot)
    ax.legend(loc='upper right', fontsize=9)

# ===== ANIMATIONSFUNKTION =====
def animate(frame):
    # Berechne den maximalen Frame für jede Bahn (basierend auf der kürzesten Simulation)
    max_frame = min(len(sol_newton.t), len(sol_gr.t), len(sol_modified.t)) - 1

    # □□□ WICHTIG: Anpassung für 15-Sekunden-Animation mit vollständigen Umläufen □□□
    # Wir beschleunigen die Animation, indem wir die Winkelgeschwindigkeit erhöhen.
    # Die physikalische Simulation läuft über viele Umläufe, aber wir "spielen sie schneller ab".
    # Skalierungsfaktor: Gesamtanzahl der Frames in der Simulation / gewünschte Frames für 15s
    scale_factor = max_frame / (15 * 3)  # 15 Sekunden bei 3 fps = 45 Frames
    current_frame_index = int(frame * scale_factor) % max_frame

    solutions = [sol_newton, sol_gr, sol_modified]

    for i, sol in enumerate(solutions):
        # Aktuelle Werte (skaliert)
        phi = sol.t[current_frame_index]
        u = sol.y[0][current_frame_index]
        r = 1 / u if u > 0 else r_max  # Vermeide Division durch Null

        # Kartesische Koordinaten
        x = r * np.cos(phi)
        y = r * np.sin(phi)

        # Update Bahn (alle Punkte bis zum aktuellen Frame)
        phi_all = sol.t[:current_frame_index+1]
        u_all = sol.y[0][:current_frame_index+1]
        r_all = np.where(u_all > 0, 1 / u_all, r_max)
        x_all = r_all * np.cos(phi_all)
        y_all = r_all * np.sin(phi_all)

        orbit_lines[i].set_data(x_all, y_all)
```

```
        particle_dots[i].set_data([x], [y])

    return orbit_lines + particle_dots

# ===== ANIMATION ERSTELLEN UND SPEICHERN =====
print("Erstelle GIF-Animation (15 Sekunden mit vollständigen Umläufen)...")

# □ Verwende 45 Frames für 15 Sekunden bei 3 fps
desired_duration_sec = 15
fps_rate = 3
total_frames = desired_duration_sec * fps_rate  # 45 Frames

ani = animation.FuncAnimation(
    fig,
    animate,
    frames=total_frames,  # Nur 45 Frames für die Animation
    interval=1000/fps_rate,  # Intervall passt zur fps-Rate
    blit=False,
    repeat=True
)

# GIF speichern mit exakt 15 Sekunden Dauer
ani.save('bahnen_vergleich_metriken.gif', writer='pillow', fps=fps_rate)

print("□ Animation 'bahnen_vergleich_metriken.gif' erfolgreich erstellt!")

plt.tight_layout()
plt.show()

print("□ Animation 'bahnen_vergleich_metriken.gif' erfolgreich erstellt!")
print("Die Animation zeigt den Fortschritt der Gravitationstheorie:")
print("1. Newton: Geschlossene Ellipse (keine Periheldrehung)")
print("2. ART: Rotierende Ellipse (Periheldrehung)")
print("3. Modifizierte Metrik: Fast identisch mit ART, aber mit subtiler Abweichung")
print("Dies illustriert, wie die reguläre Kern-Metrik die Erfolge der ART reproduziert,")
```

```
215 print("aber zusätzliche Flexibilität für Feinabstimmungen
    bietet.")
```

Listing A.12: Visualisierung Bahnenvergleich mit verschiedenen Metriken (Animation)

Hinweis zur Nutzung von KI

Die Ideen und Konzepte dieser Arbeit stammen von mir. Künstliche Intelligenz wurde unterstützend für die Textformulierung und Gleichungsformatierung eingesetzt. Die inhaltliche Verantwortung liegt bei mir. [1]

Stand: 18. September 2025
TimeStamp: https://freetsa.org/index_de.php

[1] ORCID: https://orcid.org/0009-0002-6090-3757

Literatur

[1] Ashtekar, A., & Lewandowski, J. (2006). Background independent quantum gravity: A status report. Classical and Quantum Gravity, 21(15), R53.

[2] Bardeen, J. M. (1968). Non-singular General-Relativistic Gravitational Collapse. *Proceedings of International Conference GR5*, Tbilisi, U.S.S.R.

[3] Einstein, A., Erklärung der Perihelbewegung des Merkur aus der allgemeinen Relativitätstheorie. *Sitzungsberichte der Königlich Preußischen Akademie der Wissenschaften*, 1915, 831–839.

[4] NASA/JPL Horizons System. https://ssd.jpl.nasa.gov/horizons/

[5] Kruskal, M. D. (1960). Maximal Extension of Schwarzschild Metric. *Physical Review*, 119(5), 1743–1745.

[6] Penrose, R. (1965). Gravitational collapse and space-time singularities. Physical Review Letters, 14(3), 57.

[7] Polchinski, J. (1998). String theory. Cambridge University Press.

[8] Schwarzschild, K. (1916). Über das Gravitationsfeld eines Massenpunktes nach der Einsteinschen Theorie. *Sitzungsberichte der Königlich Preußischen Akademie der Wissenschaften*, 189–196.

[9] Will, C. M. (2018). The Confrontation between General Relativity and Experiment. *Living Reviews in Relativity*, 21(1), 4.