

Numerische Differentialgleichungen auf Mannigfaltigkeiten

Eine kurze Einführung für SFZ-Projekte

Inhaltsverzeichnis

1 Geodäten auf der Kugel S^2	1
1.1 Analytische Lösung	1
1.2 Numerische Berechnung	2
1.2.1 Explizites Eulerverfahren	3
1.2.2 Implizites Eulerverfahren	4
1.2.3 Runge–Kutta-Verfahren zweiter Ordnung	5
1.2.4 Runge–Kutta-Verfahren vierter Ordnung	6
1.2.5 Adaptives Runge–Kutta–Fehlberg-Verfahren (RK45)	6
2 Python-Code	7
3 Geodätengleichungen in der Schwarzschild-Metric	12
3.1 Anfangsbedingungen	12
3.2 System erster Ordnung	13
3.3 Nächste Schritte	13

1 Geodäten auf der Kugel S^2

Die Koordinaten sind (θ, ϕ) mit $\theta \in (0, \pi)$, $\phi \in (0, 2\pi)$. Die Geodätengleichungen lauten damit

$$\begin{cases} \ddot{\theta} - \sin \theta \cos \theta \dot{\phi}^2 = 0, \\ \ddot{\phi} + 2 \cot \theta \dot{\theta} \dot{\phi} = 0. \end{cases} \quad (1)$$

Dies ist ein System nichtlinearer Differentialgleichungen zweiter Ordnung. Für eine eindeutige Lösung benötigt man Anfangsbedingungen

$$\theta(t_0) = \theta_0, \quad \dot{\theta}(t_0) = \dot{\theta}_0, \quad \phi(t_0) = \phi_0, \quad \dot{\phi}(t_0) = \dot{\phi}_0, \quad (2)$$

die die Anfangsposition und Anfangsgeschwindigkeit des Punktes auf der Kugeloberfläche festlegen.

1.1 Analytische Lösung

Die Kugelkoordinaten $(\theta(t), \phi(t))$ bestimmen die kartesische Position des Punktes durch

$$r(t) = \begin{pmatrix} \sin \theta(t) \cos \phi(t) \\ \sin \theta(t) \sin \phi(t) \\ \cos \theta(t) \end{pmatrix} \in S^2 \subset \mathbb{R}^3.$$

Umgekehrt lassen sich $\theta(t)$ und $\phi(t)$ jederzeit aus $r(t)$ zurückgewinnen. Für die Geodäten auf der Kugeloberfläche lässt sich das System (1) explizit lösen. Ausgangspunkt ist die Tatsache, dass die Bewegung eines freien Punktes auf der Kugel stets in einer Ebene durch den Ursprung erfolgt. Die Trajektorie ist daher ein Großkreis. In kartesischen Koordinaten $r(t) \in \mathbb{R}^3$ mit $\|r(t)\| = 1$ gilt für die exakte Bewegungsgleichung

$$r''(t) = -\omega^2 r(t), \quad \omega = \|r'(t_0)\|,$$

wobei $r(t_0)$ und $r'(t_0)$ aus den Anfangsbedingungen (2) bestimmt werden. Die Lösung ist damit

$$r(t) = r(t_0) \cos \omega(t - t_0) + \frac{r'(t_0)}{\omega} \sin \omega(t - t_0). \quad (3)$$

Für die Anfangsbedingungen (2) erhält man

$$r(t_0) = \begin{pmatrix} \sin \theta_0 \cos \phi_0 \\ \sin \theta_0 \sin \phi_0 \\ \cos \theta_0 \end{pmatrix}, \quad r'(t_0) = \begin{pmatrix} \dot{\theta}_0 \cos \theta_0 \cos \phi_0 - \dot{\phi}_0 \sin \theta_0 \sin \phi_0 \\ \dot{\theta}_0 \cos \theta_0 \sin \phi_0 + \dot{\phi}_0 \sin \theta_0 \cos \phi_0 \\ -\dot{\theta}_0 \sin \theta_0 \end{pmatrix}.$$

Dies ist die Gleichung eines parametrisierten Großkreises. Die zugehörigen Kugelkoordinaten ergeben sich aus

$$\theta(t) = \arccos z(t), \quad \phi(t) = \text{atan2}(y(t), x(t)).$$

Dabei bezeichnet $\text{atan2}(y, x)$ die zweistufige Arcustangens-Funktion, die den Winkel ϕ eindeutig aus den beiden Koordinaten x und y bestimmt. Sie liefert denjenigen Winkel $\phi \in (-\pi, \pi]$, für den

$$\cos \phi = \frac{x}{\sqrt{x^2 + y^2}}, \quad \sin \phi = \frac{y}{\sqrt{x^2 + y^2}},$$

und berücksichtigt im Gegensatz zum gewöhnlichen $\arctan(y/x)$ auch das Vorzeichen von x und y , sodass der richtige Quadrant bestimmt wird.

1.2 Numerische Berechnung

Um das in (1) gegebene System zweiter Ordnung numerisch zu behandeln, schreiben wir es als System erster Ordnung um. Dazu setzen wir

$$\begin{aligned} y_1 &= \theta, \\ y_2 &= \dot{y}_1 = \dot{\theta}, \\ y_3 &= \phi, \\ y_4 &= \dot{y}_3 = \dot{\phi}, \end{aligned}$$

so dass sich das erste-Ordnung-System

$$\dot{y} = \begin{pmatrix} \dot{y}_1 \\ \dot{y}_2 \\ \dot{y}_3 \\ \dot{y}_4 \end{pmatrix} = \begin{pmatrix} y_2 \\ \cos y_1 \sin y_1 y_4^2 \\ y_4 \\ -2 \cot y_1 y_2 y_4 \end{pmatrix} = f(t, y)$$

ergibt, wobei

$$y = \begin{pmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \end{pmatrix}, \quad f(t, y) = \begin{pmatrix} y_2 \\ \cos y_1 \sin y_1 y_4^2 \\ y_4 \\ -2 \cot y_1 y_2 y_4 \end{pmatrix}.$$

Die Anfangsbedingungen aus (2) übertragen sich damit direkt zu

$$y(t_0) = \begin{pmatrix} \theta_0 \\ \dot{\theta}_0 \\ \phi_0 \\ \dot{\phi}_0 \end{pmatrix}.$$

Im Folgenden kommen mehrere typische numerische Verfahren zur Anwendung, um das Anfangswertproblem

$$\dot{y} = f(t, y), \quad y(t_0) = y_0,$$

auf dem Zeitintervall $T = [0, 200]$ näherungsweise zu lösen. Die numerischen Lösungen werden anschließend mit der in Abschnitt 2.2 hergeleiteten exakten Lösung verglichen.

Dies geschieht auf zwei Arten:

- *Visueller Vergleich*: Die durch numerische Verfahren berechneten Trajektorien werden gemeinsam mit dem exakten Großkreis auf der Kugeloberfläche dargestellt.
- *Fehleranalyse*: Für jeden Zeitpunkt wird im \mathbb{R}^3 der Fehler

$$\|r_{\text{num}}(t) - r_{\text{ex}}(t)\|$$

berechnet und als Funktion der Zeit geplottet.

Auf diese Weise lässt sich die Genauigkeit der verschiedenen Integrationsmethoden direkt miteinander vergleichen.

1.2.1 Explizites Eulerverfahren

Eine der einfachsten Methoden zur numerischen Lösung von Anfangswertproblemen $\dot{y} = f(t, y)$, $y(t_0) = y_0$, ist das explizite Eulerverfahren. Für eine vorgegebene Schrittweite $h > 0$ wird die Lösung durch die Rekursion

$$y_{k+1} = y_k + h f(t_k, y_k), \quad t_{k+1} = t_k + h,$$

approximiert. Dabei wird die rechte Seite $f(t, y)$ jeweils an der bekannten Stelle (t_k, y_k) ausgewertet und dient als Vorwärts-Approximation der Tangente an die exakte Lösungskurve.

Abbildung 1 zeigt deutlich, dass die mit dem expliziten Eulerverfahren berechnete Trajektorie allmählich vom exakten Großkreis abweicht. Die numerische Bahn driftet mit der Zeit aus der Ebene der exakten Geodäte heraus und verläuft schließlich sichtbar versetzt.

Der Fehlerplot verdeutlicht diesen Drift: Die Größe $\|r_{\text{num}}(t) - r_{\text{ex}}(t)\|$ weist eine charakteristische oszillierende Struktur auf. Bei jeder „Umrundung“ der Kugel wird der Fehler kurzzeitig wieder klein, da die numerische Lösung weiterhin zwei Punkte der exakten Geodäte genau trifft: den Anfangspunkt und den gegenüberliegenden Punkt nach einem halben Umlauf. Zwischen diesen beiden Treffpunkten wächst der Abstand jedoch in jedem Umlauf etwas stärker an, sodass die maximalen Fehlerwerte stetig anwachsen. Dieser Effekt zeigt den kumulativen Einfluss der lokalen Diskretisierungsfehler des Eulerverfahrens.

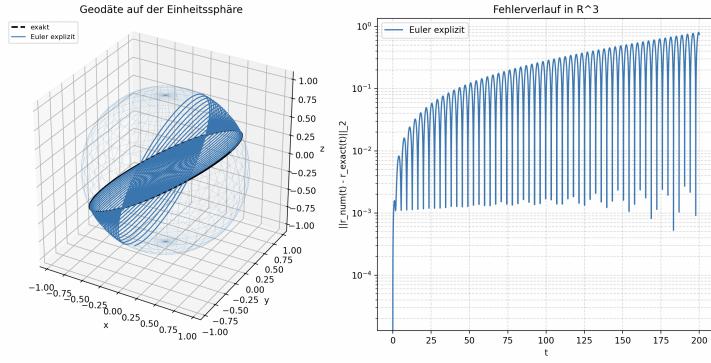


Abbildung 1: Numerische Lösung des expliziten Eulerverfahrens (Trajektorie auf der Kugeloberfläche, Fehler im \mathbb{R}^3).

1.2.2 Implizites Eulerverfahren

Eine weitere Grundmethode zur numerischen Lösung gewöhnlicher Differentialgleichungen ist das implizite Eulerverfahren. Für ein Anfangswertproblem $\dot{y} = f(t, y)$ mit $y(t_0) = y_0$ lautet die Schrittrekursion

$$y_{k+1} = y_k + h f(t_{k+1}, y_{k+1}), \quad t_{k+1} = t_k + h.$$

Im Gegensatz zum expliziten Eulerverfahren steht der neue Wert y_{k+1} hier sowohl auf der linken als auch auf der rechten Seite. Für jedes Zeitschrittintervall muss daher die Gleichung

$$y_{k+1} = y_k + h f(t_{k+1}, y_{k+1})$$

gelöst werden.

Eine einfache Möglichkeit besteht darin, diese Gleichung als Fixpunktproblem zu formulieren. Dazu definiert man die Abbildung

$$G(y) := y_k + h f(t_{k+1}, y).$$

Ein Wert y_{k+1} ist genau dann eine Lösung des impliziten Eulerschritts, wenn er ein Fixpunkt dieser Abbildung ist, also $G(y_{k+1}) = y_{k+1}$ gilt. Man erhält dadurch die Fixpunktiteration

$$y^{(m+1)} = G(y^{(m)}) = y_k + h f(t_{k+1}, y^{(m)}),$$

die mit einem geeigneten Startwert (etwa $y^{(0)} = y_k$ oder einem expliziten Eulerschritt) durchgeführt wird, bis die Differenz $\|y^{(m+1)} - y^{(m)}\|$ unter einer vorgegebenen Toleranz liegt.

Abbildung 2 zeigt, dass auch das implizite Eulerverfahren die exakte Geodäte nicht exakt nachbildet und im Verlauf vom Großkreis abdriftet. Im Vergleich zum expliziten Eulerverfahren fällt der Drift jedoch deutlich geringer aus: Die numerische Trajektorie bleibt näher an der Ebene des exakten Großkreises, und die Abweichung wächst langsamer. Dies spiegelt sich auch in der Fehleranalyse wider. Die Größe $\|r_{\text{num}}(t) - r_{\text{ex}}(t)\|$ weist eine ähnliche oszillierende Struktur auf wie beim expliziten Eulerverfahren, jedoch mit wesentlich langsamer anwachsenden Maximalwerten. Die Fixpunktiteration sorgt zwar pro Schritt für eine bessere Annäherung, kann aber die niedrige Konsistenzordnung des Verfahrens nicht vollständig kompensieren.

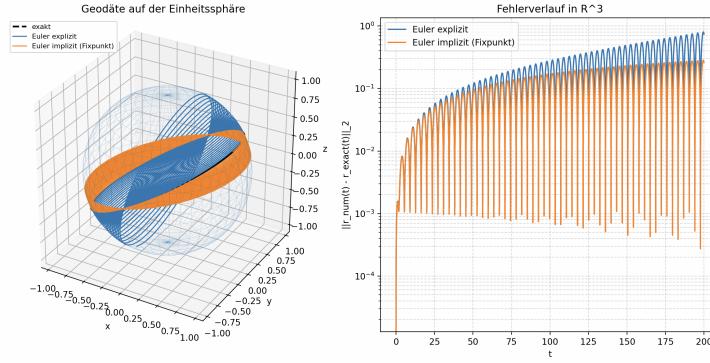


Abbildung 2: Vergleich von implizitem und explizitem Eulerverfahren mit der exakten Geodäte.

1.2.3 Runge–Kutta–Verfahren zweiter Ordnung

Zwischen dem einfachen Eulerverfahren und höherwertigen Methoden wie dem klassischen Runge–Kutta–Verfahren vierter Ordnung liegt das Runge–Kutta–Verfahren zweiter Ordnung, auch Mittelpunktsregel oder Heun–Verfahren genannt. Es verbessert den expliziten Euleransatz, indem die Steigung nicht nur am Anfang des Intervalls, sondern zusätzlich an einer Zwischenstelle ausgewertet wird.

Für einen Schritt von t_k nach $t_{k+1} = t_k + h$ berechnet man zunächst eine Vorhersage

$$k_1 = f(t_k, y_k),$$

und anschließend eine zweite Steigung an der „Mitte“ des Intervalls,

$$k_2 = f\left(t_k + \frac{h}{2}, y_k + \frac{h}{2}k_1\right).$$

Der neue Wert ergibt sich dann als

$$y_{k+1} = y_k + h k_2.$$

Durch die zusätzliche Auswertung von f erhält man eine deutlich bessere Approximation als beim expliziten Eulerverfahren, ohne dass die Methode wesentlich komplizierter wird.

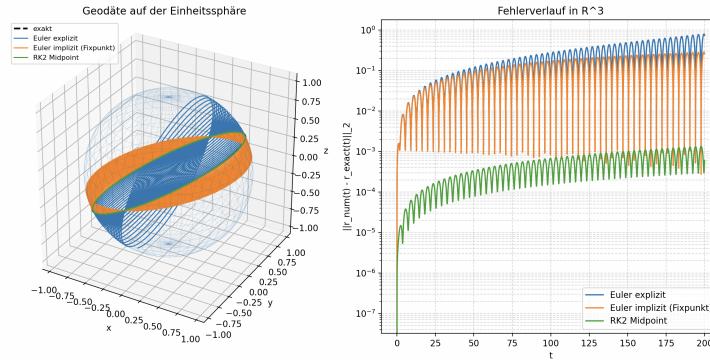


Abbildung 3: Numerische Lösung mit dem Runge–Kutta–Verfahren zweiter Ordnung im Vergleich zur exakten Geodäte.

Abbildung 3 zeigt, dass das Verfahren zweiter Ordnung die Geodäte bereits wesentlich genauer approximiert als beide Varianten des Eulerverfahrens. Zwar ist auch hier eine

leichte Abweichung vom exakten Großkreis sichtbar, doch bleibt die numerische Trajektorie deutlich näher an der Ebene der exakten Geodäte. Die Fehlerkurve zeigt weiterhin die charakteristische Oszillation über mehrere Umläufe hinweg, jedoch wächst die obere Fehlerhülle nur sehr langsam an und bleibt über das gesamte Zeitintervall hinweg unterhalb der unteren Fehlergrenze des impliziten Eulerverfahrens. Dies illustriert die deutlich höhere Genauigkeit des Verfahrens zweiter Ordnung bei gleicher Schrittweite.

1.2.4 Runge–Kutta–Verfahren vierter Ordnung

Eine weit verbreitete und äußerst zuverlässige Methode zur numerischen Integration gewöhnlicher Differentialgleichungen ist das klassische Runge–Kutta–Verfahren vierter Ordnung (RK4). Es erreicht bei nur vier Funktionsauswertungen pro Schritt eine hohe Genauigkeit und gilt als Standardmethode mit fester Schrittweite.

Für einen Schritt von t_k nach $t_{k+1} = t_k + h$ werden die Zwischensteigungen

$$k_1 = f(t_k, y_k), \quad k_2 = f(t_k + \frac{h}{2}, y_k + \frac{h}{2}k_1), \\ k_3 = f(t_k + \frac{h}{2}, y_k + \frac{h}{2}k_2), \quad k_4 = f(t_k + h, y_k + h k_3)$$

berechnet. Der neue Wert ergibt sich dann als gewichtete Kombination dieser vier Steigungen:

$$y_{k+1} = y_k + \frac{h}{6} (k_1 + 2k_2 + 2k_3 + k_4).$$

Die Gewichte sind so gewählt, dass sich für glatte rechte Seiten eine lokale Fehlerordnung von $O(h^5)$ und damit ein globaler Fehler von $O(h^4)$ ergibt.

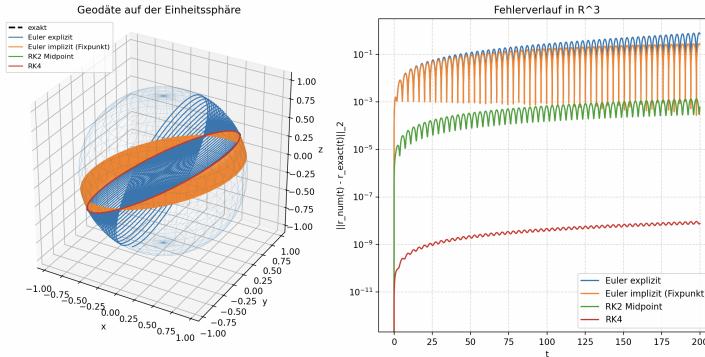


Abbildung 4: Numerische Lösung mit dem klassischen Runge–Kutta–Verfahren vierter Ordnung im Vergleich zur exakten Geodäte.

Abbildung 4 zeigt, dass das Verfahren vierter Ordnung die exakte Geodäte nahezu perfekt approximiert. Die numerische Trajektorie ist über das gesamte Zeitintervall kaum vom Großkreis zu unterscheiden. Die im Fehlerplot weiterhin vorhandenen Oszillationen haben nur noch sehr geringe Amplitude, und die maximalen Fehlerwerte bleiben um mehrere Größenordnungen kleiner als bei den Verfahren erster und zweiter Ordnung. Die deutlich verbesserte Fehlerordnung des RK4–Verfahrens wird damit anschaulich sichtbar.

1.2.5 Adaptives Runge–Kutta–Fehlberg–Verfahren (RK45)

Neben Verfahren mit fester Schrittweite werden in der Praxis häufig adaptive Verfahren verwendet, die die Schrittweite während der Integration automatisch anpassen. Ein verbreitetes Beispiel ist das Runge–Kutta–Fehlberg–Verfahren (RK45). Es basiert auf zwei eingebetteten Runge–Kutta–Formeln unterschiedlicher Ordnung: einer Formel vierter Ordnung

und einer Formel fünfter Ordnung. Beide verwenden dieselben Zwischenwerte, unterscheiden sich jedoch in der Gewichtung der Steigungen.

Seien $y_{k+1}^{(4)}$ und $y_{k+1}^{(5)}$ die beiden Näherungen für $y(t_{k+1})$, die sich aus den Formeln vierter bzw. fünfter Ordnung ergeben. Dann dient die Differenz

$$e_{k+1} = y_{k+1}^{(5)} - y_{k+1}^{(4)}$$

als Schätzwert für den lokalen Fehler. Aus diesem Fehler wird anschließend eine neue Schrittweite h_{neu} berechnet, typischerweise nach einer Regel der Form

$$h_{\text{neu}} = h \left(\frac{\text{TOL}}{\|e_{k+1}\|} \right)^{1/5},$$

wobei TOL eine vorgegebene Fehlertoleranz ist. Schritte, deren geschätzter Fehler zu groß ist, werden mit kleinerer Schrittweite wiederholt; ist der Fehler sehr klein, wird die Schrittweite vergrößert.

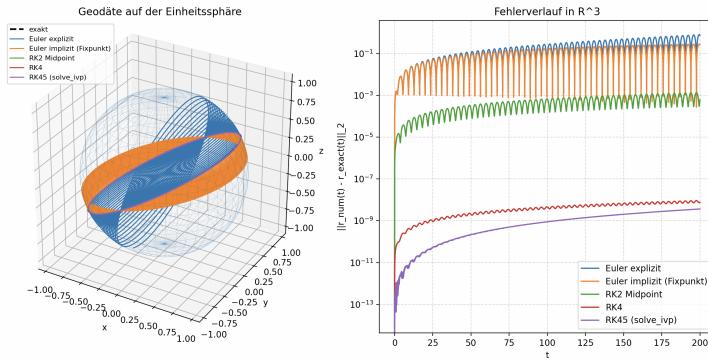


Abbildung 5: Numerische Lösung mit dem adaptiven Runge–Kutta–Fehlberg-Verfahren (RK45) im Vergleich zur exakten Geodäte.

Abbildung 5 zeigt, dass das adaptive RK45-Verfahren die exakte Geodäte über moderate Zeiten hinweg extrem präzise approximiert. Aufgrund der automatischen Schrittweitensteuerung werden in Bereichen geringer Krümmung große und in Bereichen stärkerer Krümmung kleine Schritte gemacht, wodurch der lokale Fehler effektiv kontrolliert wird. Für die im Plot verwendeten Toleranzen $\text{rtol} = \text{atol} = 10^{-12}$ liegt der Fehler über das gezeigte Intervall sogar unterhalb des Fehlers des festen RK4-Verfahrens.

Bei sehr langen Integrationszeiten zeigt sich jedoch ein Unterschied: Während RK4 mit fester kleiner Schrittweite einen sehr gleichmäßigen globalen Fehler erzeugt, wächst der globale Fehler des adaptiven RK45 aufgrund der dynamisch größeren Schrittweiten schneller an und kann auf langen Intervallen schließlich oberhalb des RK4-Fehlerniveaus liegen. Dafür benötigt RK45 deutlich weniger Funktionsauswertungen und ist damit in der Praxis wesentlich effizienter.

2 Python-Code

Im Folgenden ist das vollständige Python-Programm abgedruckt, mit dem die verschiedenen numerischen Verfahren miteinander verglichen werden. Man fühlt sich dabei fast an frühere Zeiten erinnert, in denen man Programme aus Computerzeitschriften Zeile für Zeile abgetippt hat. Hier soll das Listing jedoch weniger zum Abschreiben dienen, sondern

vor allem als archivierte Referenz. Für eine Veröffentlichung auf GitHub ist mir dieser Code etwas zu stark „zusammengehackt“ und eher als experimentelle Arbeitsversion zu verstehen. Im LaTeX-Dokument ist er dagegen gut aufgehoben und kann bei Bedarf leicht reproduziert werden.

Für die adaptive Methode RK45 wird `scipy` verwendet; die übrigen Verfahren (explizites Eulerverfahren, implizites Eulerverfahren, Runge–Kutta zweiter und vierter Ordnung) lassen sich problemlos direkt in Python implementieren. Zur Visualisierung wird `matplotlib` eingesetzt. Mit MATLAB wäre ein solches Programm vermutlich ebenso schnell geschrieben, doch bietet Python hier den Vorteil der leichten Erweiterbarkeit und offenen Bibliothekslandschaft.

Listing 1: Python-Programm zur numerischen Berechnung der Geodäte

```

1 import numpy as np
2 from math import sin, cos
3 import matplotlib.pyplot as plt
4 from scipy.integrate import solve_ivp
5
6 # -----
7 # ODE: Geodäten auf der Einheitssphäre
8 # y = [theta, theta_dot, phi, phi_dot]
9 # -----
10
11
12 def geodesic_rhs(t, y):
13     theta, theta_dot, phi, phi_dot = y
14     s = np.sin(theta)
15     c = np.cos(theta)
16
17     dtheta_dt = theta_dot
18     dphi_dt = phi_dot
19
20     # theta'' = cos(theta) sin(theta) (phi')^2
21     dtheta_dot_dt = c * s * (phi_dot**2)
22
23     # phi'' = -2 cot(theta) theta' phi'
24     if abs(s) < 1e-8:
25         dphi_dot_dt = 0.0 # Koordinatensingularität "abfangen"
26     else:
27         cot_theta = c / s
28         dphi_dot_dt = -2.0 * cot_theta * theta_dot * phi_dot
29
30     return np.array([dtheta_dt, dtheta_dot_dt, dphi_dt, dphi_dot_dt])
31
32
33 def spherical_to_cartesian(theta, phi):
34     x = np.sin(theta) * np.cos(phi)
35     y = np.sin(theta) * np.sin(phi)
36     z = np.cos(theta)
37     return x, y, z
38
39
40 # -----
41 # Exakte Lösung in 3D (Großkreis)
42 # -----
43
44
45 def initial_cartesian_and_velocity(theta0, theta_dot0, phi0, phi_dot0):
46     s = np.sin(theta0)
47     c = np.cos(theta0)
```

```

48     cp = np.cos(phi0)
49     sp = np.sin(phi0)
50
51     r0 = np.array([s * cp, s * sp, c])
52     dr_dtheta = np.array([c * cp, c * sp, -s])
53     dr_dphi = np.array([-s * sp, s * cp, 0.0])
54     v0 = theta_dot0 * dr_dtheta + phi_dot0 * dr_dphi
55     return r0, v0
56
57
58 def exact_geodesic_3d(t_array, y0):
59     theta0, theta_dot0, phi0, phi_dot0 = y0
60     r0, v0 = initial_cartesian_and_velocity(theta0, theta_dot0, phi0, phi_dot0)
61     omega = np.linalg.norm(v0)
62     if omega == 0.0:
63         r_exact = np.tile(r0, (len(t_array), 1))
64         return r_exact[:, 0], r_exact[:, 1], r_exact[:, 2]
65
66     # r(t) = r0 cos( t ) + (v0 /) sin( t )
67     t_array = np.asarray(t_array)
68     cos_term = np.cos(omega * t_array)[:, None]
69     sin_term = np.sin(omega * t_array)[:, None]
70
71     r_exact = r0 * cos_term + (v0 / omega) * sin_term
72     return r_exact[:, 0], r_exact[:, 1], r_exact[:, 2]
73
74
75 # -----
76 # Allgemeiner Integrator mit fester Schrittweite
77 # -----
78
79
80 def integrate_fixed_step(f, t0, y0, h, n_steps, stepper):
81     t = np.empty(n_steps + 1)
82     y = np.empty((n_steps + 1, len(y0)))
83     t[0] = t0
84     y[0] = np.array(y0, dtype=float)
85
86     for k in range(n_steps):
87         t[k + 1], y[k + 1] = stepper(f, t[k], y[k], h)
88
89     return t, y
90
91
92 def euler_explicit_step(f, t, y, h):
93     return t + h, y + h * f(t, y)
94
95
96 def euler_implicit_step(f, t, y, h, tol=1e-12, max_iter=50):
97     t_new = t + h
98
99     y_old = y.copy()
100    y_new = y + h * f(t_new, y_old) # besserer Start als y_old
101
102    for k in range(max_iter):
103        y_prev = y_new.copy()
104        y_new = y + h * f(t_new, y_prev)
105
106        if np.linalg.norm(y_new - y_prev, ord=np.inf) < tol:
107            return t_new, y_new
108

```

```

109     print("Warnung: Fixpunktiteration hat nicht konvergiert.")
110     return t_new, y_new # trotzdem zurückgeben
111
112
113 def rk2_midpoint_step(f, t, y, h):
114     k1 = f(t, y)
115     k2 = f(t + 0.5 * h, y + 0.5 * h * k1)
116     return t + h, y + h * k2
117
118
119 def rk4_step(f, t, y, h):
120     k1 = f(t, y)
121     k2 = f(t + 0.5 * h, y + 0.5 * h * k1)
122     k3 = f(t + 0.5 * h, y + 0.5 * h * k2)
123     k4 = f(t + h, y + h * k3)
124     return t + h, y + (h / 6.0) * (k1 + 2 * k2 + 2 * k3 + k4)
125
126
127 def main():
128     # Anfangsbedingungen
129     theta0 = np.pi / 3 # 60°
130     phi0 = 0.0
131     theta_dot0 = 0.0
132     phi_dot0 = 1.0
133     y0 = np.array([theta0, theta_dot0, phi0, phi_dot0])
134
135     # Integrationsintervall und Schrittweite
136     t0 = 0.0
137     T = 4000.0
138     h = 0.01
139     n_steps = int((T - t0) / h)
140
141     # Feste-Schrittweiten-Methoden
142     methods = {
143         # "Euler explizit": euler_explicit_step,
144         "Euler\u2225Implizit\u2225(Fixpunkt)": lambda f, t, y, h: euler_implicit_step(
145             f, t, y, h
146         ),
147         "RK2\u2225Midpoint": rk2_midpoint_step,
148         "RK4": rk4_step,
149     }
150
151     trajectories = {}
152     errors = {}
153
154     # Exakte Lösung auf dem gemeinsamen Gitter
155     t_grid = np.linspace(t0, T, n_steps + 1)
156     x_ex, y_ex, z_ex = exact_geodesic_3d(t_grid, y0)
157     r_exact = np.stack((x_ex, y_ex, z_ex), axis=1)
158
159     # Feste Schrittweite
160     for name, stepper in methods.items():
161         t, y = integrate_fixed_step(geodesic_rhs, t0, y0, h, n_steps, stepper)
162         theta = y[:, 0]
163         phi = y[:, 2]
164         x, yy, z = spherical_to_cartesian(theta, phi)
165         trajectories[name] = (x, yy, z)
166
167         r_num = np.stack((x, yy, z), axis=1)
168         err = np.linalg.norm(r_num - r_exact, axis=1)
169         print(err)

```

```

170     errors[name] = err
171
172 # Referenz mit solve_ivp (adaptives RK45), aber gleiche Ausgabezeiten
173 sol = solve_ivp(
174     geodesic_rhs,
175     (t0, T),
176     y0,
177     t_eval=t_grid,
178     method="RK45",
179     rtol=1e-12,
180     atol=1e-12,
181 )
182 theta_ref = sol.y[0]
183 phi_ref = sol.y[2]
184 x_ref, y_ref, z_ref = spherical_to_cartesian(theta_ref, phi_ref)
185 trajectories["RK45\u2225(solve_ivp)"] = (x_ref, y_ref, z_ref)
186
187 r_ref = np.stack((x_ref, y_ref, z_ref), axis=1)
188 errors["RK45\u2225(solve_ivp)"] = np.linalg.norm(r_ref - r_exact, axis=1)
189
190 # ----- Plot 1: Trajektorien auf der Kugel -----
191 fig = plt.figure(figsize=(12, 6))
192 ax3d = fig.add_subplot(1, 2, 1, projection="3d")
193
194 # Drahtgitter-Kugel
195 u = np.linspace(0, 2 * np.pi, 50)
196 v = np.linspace(0, np.pi, 25)
197 U, V = np.meshgrid(u, v)
198 X = np.sin(V) * np.cos(U)
199 Y = np.sin(V) * np.sin(U)
200 Z = np.cos(V)
201 ax3d.plot_wireframe(X, Y, Z, linewidth=0.3, alpha=0.3)
202
203 # Exakte Trajektorie
204 ax3d.plot(x_ex, y_ex, z_ex, "k--", linewidth=2, label="exakt")
205
206 # Numerische Trajektorien
207 for name, (x, yy, z) in trajectories.items():
208     ax3d.plot(x, yy, z, label=name, linewidth=1.2)
209
210 ax3d.set_box_aspect([1, 1, 1])
211 ax3d.set_xlabel("x")
212 ax3d.set_ylabel("y")
213 ax3d.set_zlabel("z")
214 ax3d.set_title("Geodäte\u2225auf\u2225der\u2225Einheitssphäre")
215 ax3d.legend(loc="upper left", fontsize=8)
216
217 for setter in (ax3d.set_xlim, ax3d.set_ylim, ax3d.set_zlim):
218     setter([-1.1, 1.1])
219
220 # ----- Plot 2: Fehlerverlauf -----
221 ax_err = fig.add_subplot(1, 2, 2)
222 for name, err in errors.items():
223     ax_err.semilogy(t_grid, err, label=name)
224
225 ax_err.set_xlabel("t")
226 ax_err.set_ylabel("||r_num(t)-r_exact(t)||_2")
227 ax_err.set_title("Fehlerverlauf\u2225in\u2225R^3")
228 ax_err.legend()
229 ax_err.grid(True, which="both", linestyle="--", alpha=0.5)
230

```

```

231     plt.tight_layout()
232     plt.show()
233
234
235 if __name__ == "__main__":
236     main()

```

3 Geodätengleichungen in der Schwarzschild-Metric

Wir betrachten die Geodäten im Schwarzschild-Raumzeitmodell. Alle Koordinaten

$$t = t(\lambda), \quad r = r(\lambda), \quad \theta = \theta(\lambda), \quad \phi = \phi(\lambda)$$

werden dabei als Funktionen eines affinen Parameters λ aufgefasst. Ein Punkt über einer Variablen bedeutet die Ableitung nach λ , also

$$\dot{r} = \frac{dr}{d\lambda}, \quad \ddot{r} = \frac{d^2r}{d\lambda^2}, \quad \text{etc.}$$

Die Geodätengleichungen lauten:

$$\begin{aligned} \ddot{t} + \frac{2M}{r(r-2M)} \dot{t} \dot{r} &= 0 \\ \ddot{r} + \frac{M(r-2M)}{r^3} \dot{t}^2 - \frac{M}{r(r-2M)} \dot{r}^2 - (r-2M) \dot{\theta}^2 - (r-2M) \sin^2 \theta \dot{\phi}^2 &= 0, \\ \ddot{\theta} + \frac{2}{r} \dot{r} \dot{\theta} - \sin \theta \cos \theta \dot{\phi}^2 &= 0, \\ \ddot{\phi} + \frac{2}{r} \dot{r} \dot{\phi} + 2 \cot \theta \dot{\theta} \dot{\phi} &= 0. \end{aligned}$$

3.1 Anfangsbedingungen

Zur eindeutigen Lösung werden Werte bei $\lambda = 0$ gewählt:

$$t(0) = t_0, \quad r(0) = r_0, \quad \theta(0) = \theta_0, \quad \phi(0) = \phi_0,$$

sowie

$$\dot{t}(0) = v_{t,0}, \quad \dot{r}(0) = v_{r,0}, \quad \dot{\theta}(0) = v_{\theta,0}, \quad \dot{\phi}(0) = v_{\phi,0}.$$

Bei der Herleitung der Geodätengleichungen erhält man als Nebenresultat, dass der Ausdruck

$$-\left(1 - \frac{2M}{r}\right) \dot{t}^2 + \left(1 - \frac{2M}{r}\right)^{-1} \dot{r}^2 + r^2 \dot{\theta}^2 + r^2 \sin^2 \theta \dot{\phi}^2$$

entlang jeder Lösung identisch konstant bleibt. Die auftretende Konstante wird durch die gewählten Anfangswerte festgelegt und ändert sich anschließend nicht mehr entlang der Kurve.

3.2 System erster Ordnung

Um das gegebene System zweiter Ordnung numerisch zu behandeln, schreiben wir es als System erster Ordnung um. Dazu setzen wir

$$\begin{aligned} y_1 &= r, \\ y_2 &= \dot{y}_1 = \dot{r}, \\ y_3 &= t, \\ y_4 &= \dot{y}_3 = \dot{t}, \\ y_5 &= \theta, \\ y_6 &= \dot{y}_5 = \dot{\theta}, \\ y_7 &= \phi, \\ y_8 &= \dot{y}_7 = \dot{\phi}. \end{aligned}$$

Dann ergibt sich ein System erster Ordnung

$$\dot{y} = f(\lambda, y)$$

mit

$$f(\lambda, y) = \begin{pmatrix} y_2 \\ -\frac{M(y_1 - 2M)}{y_1^3} y_4^2 + \frac{M}{y_1(y_1 - 2M)} y_2^2 + (y_1 - 2M) y_6^2 + (y_1 - 2M) \sin^2(y_5) y_8^2 \\ y_4 \\ -\frac{2M}{y_1(y_1 - 2M)} y_2 y_4 \\ y_6 \\ -\frac{2}{y_1} y_2 y_6 + \sin(y_5) \cos(y_5) y_8^2 \\ y_8 \\ -\frac{2}{y_1} y_2 y_8 - 2 \cot(y_5) y_6 y_8 \end{pmatrix}.$$

3.3 Nächste Schritte

Das resultierende System erster Ordnung kann nun mit numerischen Verfahren wie dem klassischen Runge–Kutta-Verfahren (RK4) oder einem adaptiven Verfahren (RK45) gelöst werden. Die Situation ist damit ähnlich wie bei den zuvor betrachteten Geodäten auf der Kugeloberfläche S^2 : Wir erhalten eine parametrisierte Kurve $\lambda \mapsto y(\lambda)$, deren Komponenten (r, θ, ϕ) den Ort relativ zum schwarzen Loch beschreiben.

Offen ist noch die Frage, wie man die numerische Lösung am sinnvollsten visualisieren kann. Hierfür bieten sich verschiedene Ansätze an:

- (A) *Bewegung in der Äquatorebene.* Setzt man $\theta(0) = \frac{\pi}{2}$ und $\dot{\theta}(0) = 0$, so bleibt die Bewegung in der Ebene. Die Bahn kann dann über

$$x(\lambda) = r(\lambda) \cos \phi(\lambda), \quad y(\lambda) = r(\lambda) \sin \phi(\lambda)$$

als Kurve in der Ebene dargestellt werden.

- (B) *Radiale Profile.* Man kann r gegen den Parameter λ oder gegen t auftragen, um den radialen Verlauf der Bewegung zu visualisieren. Auch Darstellungen von $r(\phi)$ sind möglich.
- (C) *Vollständige dreidimensionale Darstellung.* Für allgemeine Anfangswerte kann die Bewegung in drei Dimensionen visualisiert werden, indem man aus (r, θ, ϕ) die kartesischen Koordinaten berechnet. Dies ist aufwendiger, erlaubt aber die Betrachtung beliebiger räumlicher Geodäten.