Allgemeine Relativitätstheorie mit dem Computer

VORLESUNGSREIHE
JOHANN WOLFGANG GOETHE UNIVERSITÄT
04. MAI, 2023

MATTHIAS HANAUSKE

FRANKFURT INSTITUTE FOR ADVANCED STUDIES
JOHANN WOLFGANG GOETHE UNIVERSITÄT
INSTITUT FÜR THEORETISCHE PHYSIK
ARBEITSGRUPPE RELATIVISTISCHE ASTROPHYSIK
D-60438 FRANKFURT AM MAIN
GERMANY

4. Vorlesung

Jupyter Notebook

Radialer Fall eines Probekörpers in ein nicht-rotierendes Schwarzes Loch

Allgemeine Relativitätstheorie mit dem Computer

General Theory of Relativity on the Computer

Vorlesung gehalten an der J.W.Goethe-Universität in Frankfurt am Main (Sommersemester 2021)

von Dr.phil.nat. Dr.rer.pol. Matthias Hanauske

Frankfurt am Main 04.04.2021

Erster Vorlesungsteil: Allgemeine Relativitätstheorie mit Python ¶

Bewegung eines Probekörpers um ein nicht-rotierendes schwarzes Loch in der Ebene

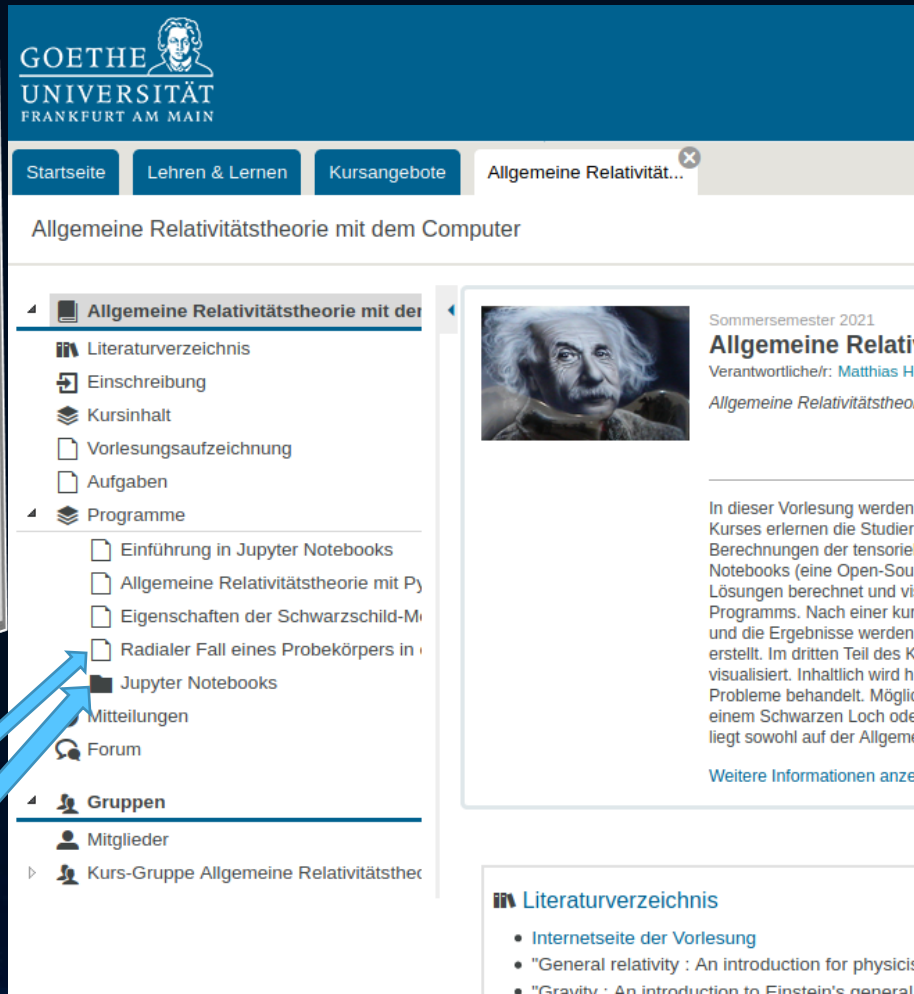
Teil I: Radialer Fall eines Probekörpers

Im Folgenden wird die Geodätengleichung in vorgegebener Schwarzschild Raumzeit betrachtet. Die Geodätengleichung beschreibt wie sich ein Probekörper (Masse Probekörper $m << M$ Masse schwarzes Loch) im Raum bewegt und sagt voraus, dass diese Bewegung sich stehts entlang der kürzesten Kurve, in der durch die Metrik beschriebenen gekrümmten Raumzeit, vollzieht. Zunächst wird das Python Modul "EinsteinPy" eingebunden, welches auf dem Modul SymPy basiert und symbolische Berechnungen in der Allgemeinen Relativitätstheorie relativ einfach möglich macht.

Definition der Koordinaten und der kovarianten Raumzeit-Metrik der Schwarzschildmetrik:

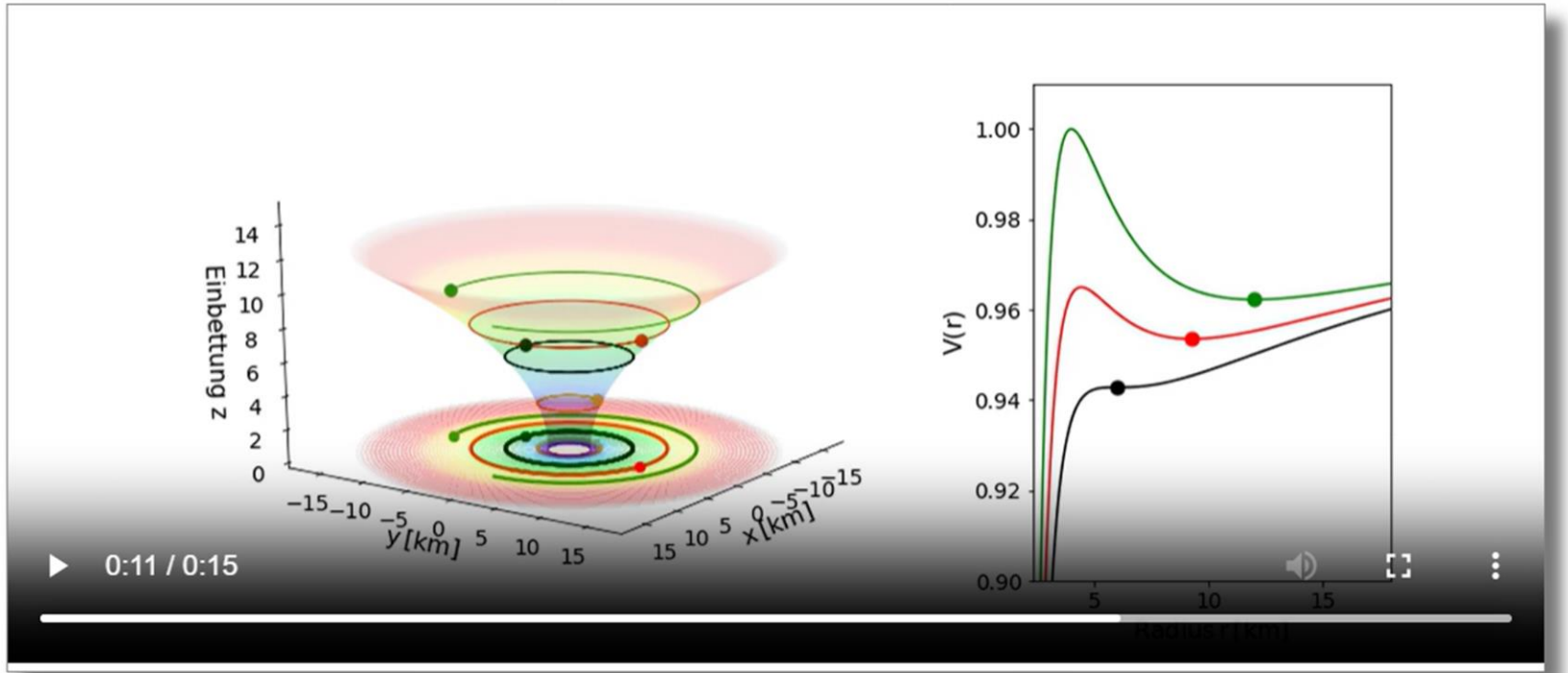
$$\begin{pmatrix} 1-\frac{2M}{r} & 0 & 0 & 0 \\ & \frac{1}{} & 0 & 0 \end{pmatrix}$$

Auf der OLAT Seite des Kurses finden Sie die Jupyter Notebooks zum Ansehen und zum herunterladen



Die innerste stabile kreisförmige Bahnbewegung (der ISCO: Innermost Stable Circular Orbit) und die Photonensphäre

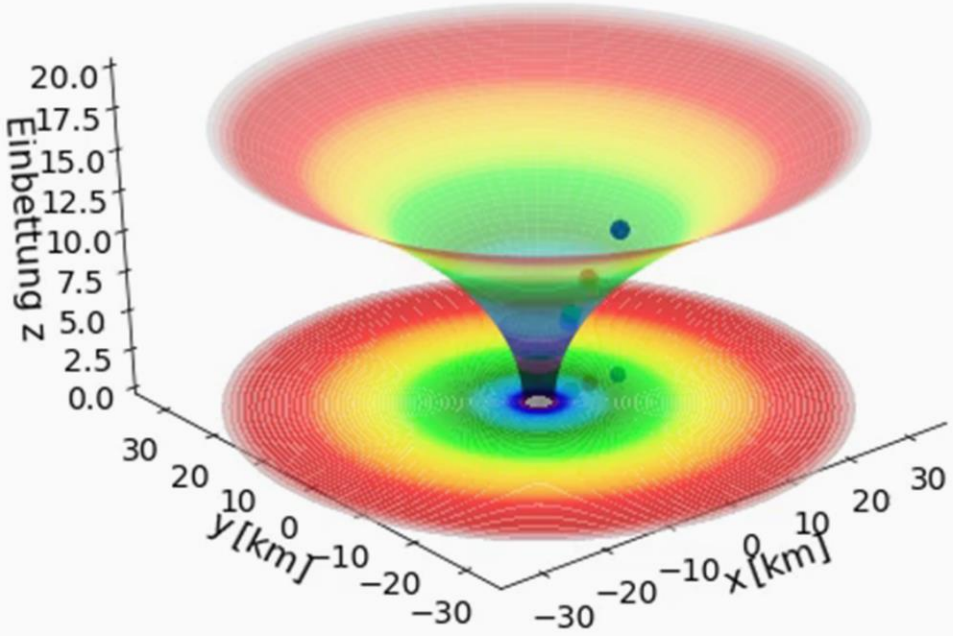
Die linke Seite der unteren Animation verdeutlicht die Trajektorien des ISCO und der Photonensphäre in einem eingebetteten Raumzeit-Diagramm der Schwarzschild-Metrik. Kreisförmige Bewegungen von massiven Probekörpern um ein nicht rotierendes schwarzes Loch können dem Loch nicht beliebig nahekommen. Die Grenze der Stabilität von kreisförmigen Trajektorien nennt man ISCO (siehe rechtes Panel dieser Vorlesung). Die linke untere Abbildung zeigt die simulierten Bahnen dreier kreisförmige Bewegungen, welche in unterschiedlichen Abstand um das schwarze Loch kreisen, wobei die Masse des schwarzen Loches auf $M = 1$ gesetzt wurde. Der grüne Probekörper kreist mit einem Abstand von $r = 12$ km, der rote Körper von $r = 9.25$ km und der schwarze Körper von $r = 6$ km um das schwarze Loch. Zusätzlich sind für diese kreisförmigen Orbits auf der rechten Seite der unteren Abbildung die zugehörigen effektiven Potentiale dargestellt. Da es sich um kreisförmige Bewegungen handelt befinden sich alle drei Probekörper im Minimum des zugehörigen Potentials bzw. der schwarze Orbit im Sattelpunkt des Potentials. Stabile kreisförmige Bewegungen um ein schwarzes Loch (nicht-rotierend) sind somit nur bis zu einem Abstand von $r = 6M$ möglich. Das gilt jedoch nicht für masselose Probekörper (z.B. Photonen). Die orange Kurve auf der linken Seite der unteren Abbildung zeigt die Bewegung eines masselosen Testteilchen im Abstand $r = 3$ km. Bei diesem Abstand können Lichtteilchen kreisförmig um das schwarze Loch rotieren und man nennt die entsprechende Kugelschale bei $r = 3M$ auch deshalb die Photonensphäre eines schwarzen Loches.

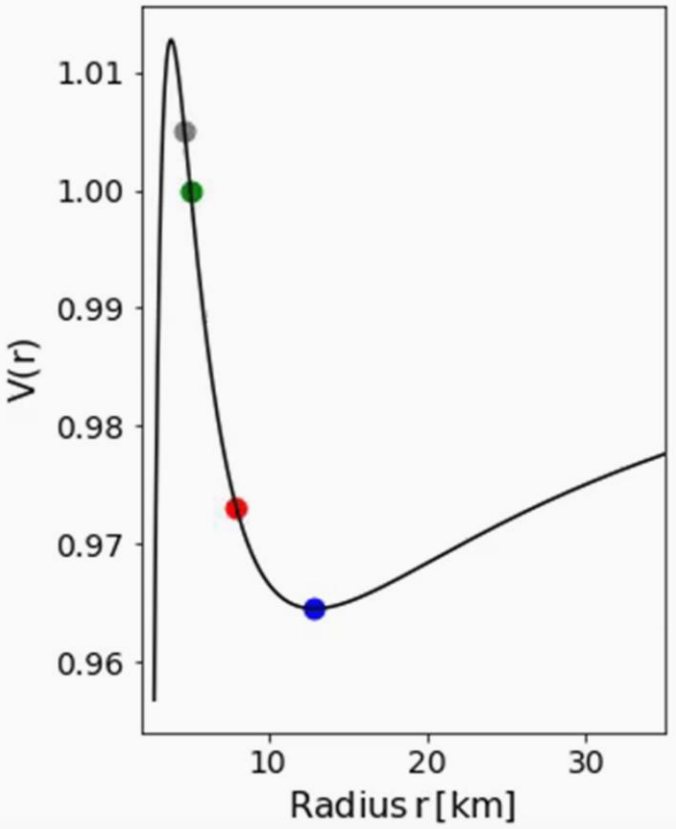


Klassifizierung der möglichen Bahnbewegungen eines Probekörpers um ein nicht-rotierendes schwarzes Loch

Neben den gebundenen kreisförmigen (blau) und elliptischen (rot) Bahnen, den parabolischen (grün) und hyperbolischen (grau) Bahnverläufen ist auch eine durch das schwarze Loch eingefangene Bahn (schwarz: capture orbit) möglich

Animation wurde im Python Jupyter Notebook *„Klassifizierung unterschiedlicher Bahnbewegungen um ein nicht-rotierendes Schwarzes Loch“* erstellt





Systeme von gekoppelten Differentialgleichungen

Wir betrachten zunächst das numerische Lösen eines Systems von m-gekoppelten Differentialgleichungen (DGLs) erster Ordnung der Form

$$\begin{aligned}
\dot{y}_1(t) &= \frac{dy_1}{dt} = f_1(t, y_1, y_2, \ldots, y_m) \\
\dot{y}_2(t) &= \frac{dy_2}{dt} = f_2(t, y_1, y_2, \ldots, y_m) \\
\dot{y}_3(t) &= \ldots = \\
&\ldots = \ldots \\
\dot{y}_m(t) &= \frac{dy_m}{dt} = f_m(t, y_1, y_2, \ldots, y_m) \quad ,
\end{aligned}$$

wobei die zeitliche Entwicklung der Vektorfunktion $\vec{y}(t) = (y_1(t), y_2(t), \ldots, y_m(t))$ in den Grenzen $a \leq t \leq b$ gesucht wird. Die m-Funktionen $f_i(t, y_1, y_2, \ldots, y_m)$, $i \in [1, 2, \ldots, m]$ bestimmen das System der DGLs und somit das Verhalten der gesuchten Funktion $\vec{y}(t)$. Es wird hierbei vorausgesetzt, dass die Funktionen $f_i(t, y_1, y_2, \ldots, y_m)$ auf einer Teilmenge $\mathcal{D}$ ($\mathbb{R}^{m+1} \supseteq \mathcal{D}$) kontinuierlich definiert sind und das so definierte Anfangswertproblem "well-posed" ist und eine eindeutige Lösung $\vec{y}(t)$ existiert. Bei gegebener Anfangskonfiguration

$$y_1(a) = \alpha_1 \, , \; y_2(a) = \alpha_2 \, , \; \ldots \; , \; y_m(a) = \alpha_m$$

ist es dann numerisch möglich das System von gekoppelten DGLs zu lösen.

Die Geodätengleichung

Ein System gekoppelter nicht-linearer Differentialgleichungen zweiter Ordnung?

, wobei wir wieder ein sphärisches Koordinatensystem benutzen ($x^\mu = (t, r, \theta, \phi)$). Die Geodätengleichung stellt ein System gekoppelter nichtlinearer Differentialgleichungen dar

$$\frac{d^2t}{d\lambda^2} = -\Gamma^0_{\nu\rho}\frac{dx^\nu}{d\lambda}\frac{dx^\rho}{d\lambda}$$
$$\frac{d^2r}{d\lambda^2} = -\Gamma^1_{\nu\rho}\frac{dx^\nu}{d\lambda}\frac{dx^\rho}{d\lambda}$$
$$\frac{d^2\theta}{d\lambda^2} = -\Gamma^2_{\nu\rho}\frac{dx^\nu}{d\lambda}\frac{dx^\rho}{d\lambda}$$
$$\frac{d^2\phi}{d\lambda^2} = -\Gamma^3_{\nu\rho}\frac{dx^\nu}{d\lambda}\frac{dx^\rho}{d\lambda} \quad ,$$

wobei λ ein affiner Parameter (z.B. die Eigenzeit τ), t, r, θ und ϕ die Koordinaten und $\Gamma^\mu_{\nu\rho}$ die Christoffel Symbole zweiter Art darstellen.

Literatur:

1. *General relativity : An introduction for physicists* von M. P. Hobson, G. P. Efstathiou und A. N. Lasenby
2. *Gravity : An introduction to Einstein's general relativity* von James B. Hartle
3. *Allgemeine Relativitätstheorie* von Torsten Fließbach
4. *Relativistic hydrodynamics* von Luciano Rezzolla und Olindo Zanotti

Während der Bewegung erhaltene Größen

Energie und Drehimpuls des Probekörpers

Hobson, G. P. Efstathiou and A. N. Lasenby), dass sich die erste und vierte Gleichung des Systems der Differentialgleichungen der Geodätengleichung in die folgenden Gleichungen umschreiben lässt:

$$\text{1. Gleichung: } \frac{d}{d\tau}\left[\left(1-\frac{2M}{r}\right)\frac{dt}{d\tau}\right]=0$$

$$\rightarrow \left(1-\frac{2M}{r}\right)\frac{dt}{d\tau}=e=\text{const}$$

$$\text{4. Gleichung: } \frac{d}{d\tau}\left(r^2\sin^2(\theta)\frac{d\phi}{d\tau}\right)=0$$

$$\rightarrow r^2\sin^2(\theta)\frac{d\phi}{d\tau}=l=\text{const} \quad ,$$

wobei die während der Bewegungen erhaltenen Größen e (Teilchenenergie pro Masse, $e=\frac{E}{m}$) und l (Drehimpuls pro Masse m, $l=\frac{L}{m}$) sich aus der Definition des Viererimpulses $p_\mu = m\,u_\mu = m\,g_{\mu\nu}\frac{dx^\nu}{d\tau} = m\left(g_{tt}\frac{dt}{d\tau}, g_{rr}\frac{dr}{d\tau}, g_{\theta\theta}\frac{d\theta}{d\tau}, g_{\phi\phi}\frac{d\phi}{d\tau}\right)$ ergeben (Voraussetzung: diagonale Form der Metrik). Für die Schwarzschildmetrik erhält man:

$$p_t = p_0 = m\frac{dx_0}{d\tau} = mg_{00}\frac{dx^0}{d\tau} = mg_{00}\frac{dt}{d\tau} = m\left(1-\frac{2M}{r}\right)\frac{dt}{d\tau} = m\,e$$

$$p_\phi = p_3 = m\frac{dx_3}{d\tau} = mg_{33}\frac{dx^3}{d\tau} = mg_{33}\frac{d\phi}{d\tau} = -m\left(r^2\sin^2(\theta)\right)\frac{d\phi}{d\tau} = -m\,l$$

Literatur:

1. *General relativity : An introduction for physicists* von M. P. Hobson, G. P. Efstathiou und A. N. Lasenby
2. *Gravity : An introduction to Einstein's general relativity* von James B. Hartle
3. *Allgemeine Relativitätstheorie* von Torsten Fließbach
4. *Relativistic hydrodynamics* von Luciano Rezzolla und Olindo Zanotti

Das effektive Potential V(r,M,l)

Die Klassifikation möglicher Bahnen von Probekörpern in vorgegebener Schwarzschild-Raumzeit kann mittels eines definierten effektiven Potentials illustriert werden. Dieses Potential hängt von dem, bei der Bewegung erhaltenen Drehimpuls pro Masse m ab. Die im Zentralfeld möglichen Bewegungen werden mittels zweier erhaltener Größen (l: Drehimpuls pro Masse m und e: Energie pro Masse) charakterisiert. Die Definition des effektiven Potential erfolgt mittels der radialen, 2. Geodätengleichung. So definieren z.B. die Literaturangaben 1-3 (siehe Angaben oben) das effektive Potentail wie folgt:

$$\text{2. Gleichung: } \rightarrow \frac{1}{2}\left(\frac{dr}{d\tau}\right)^2 + V(r, M, l) = \frac{1}{2}\left(e^2 - 1\right)$$

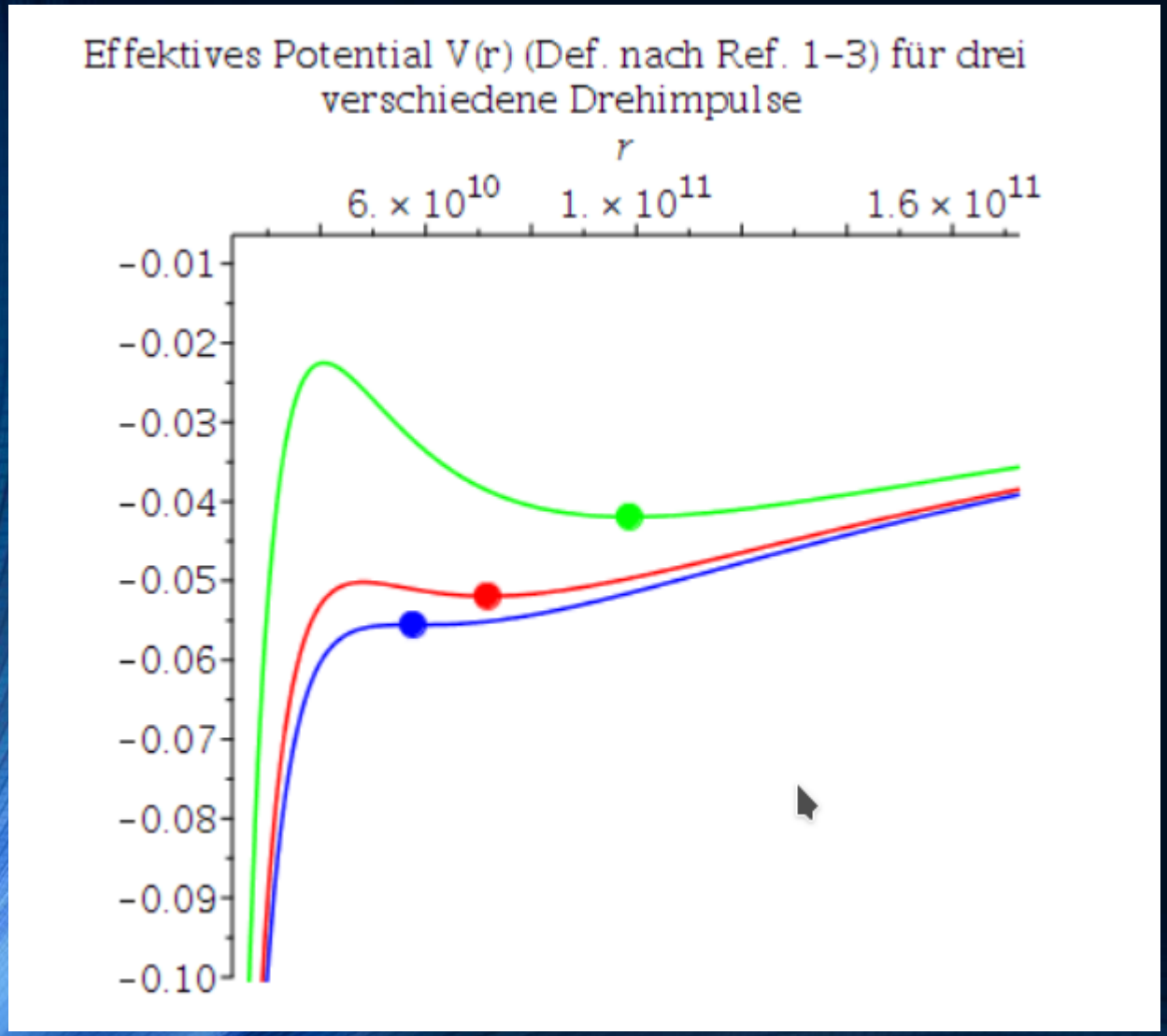
$$\text{wobei: } \quad V(r, M, l) = \frac{l^2}{2r^2}\left(1 - \frac{2M}{r}\right) - \frac{M}{r} \quad .$$

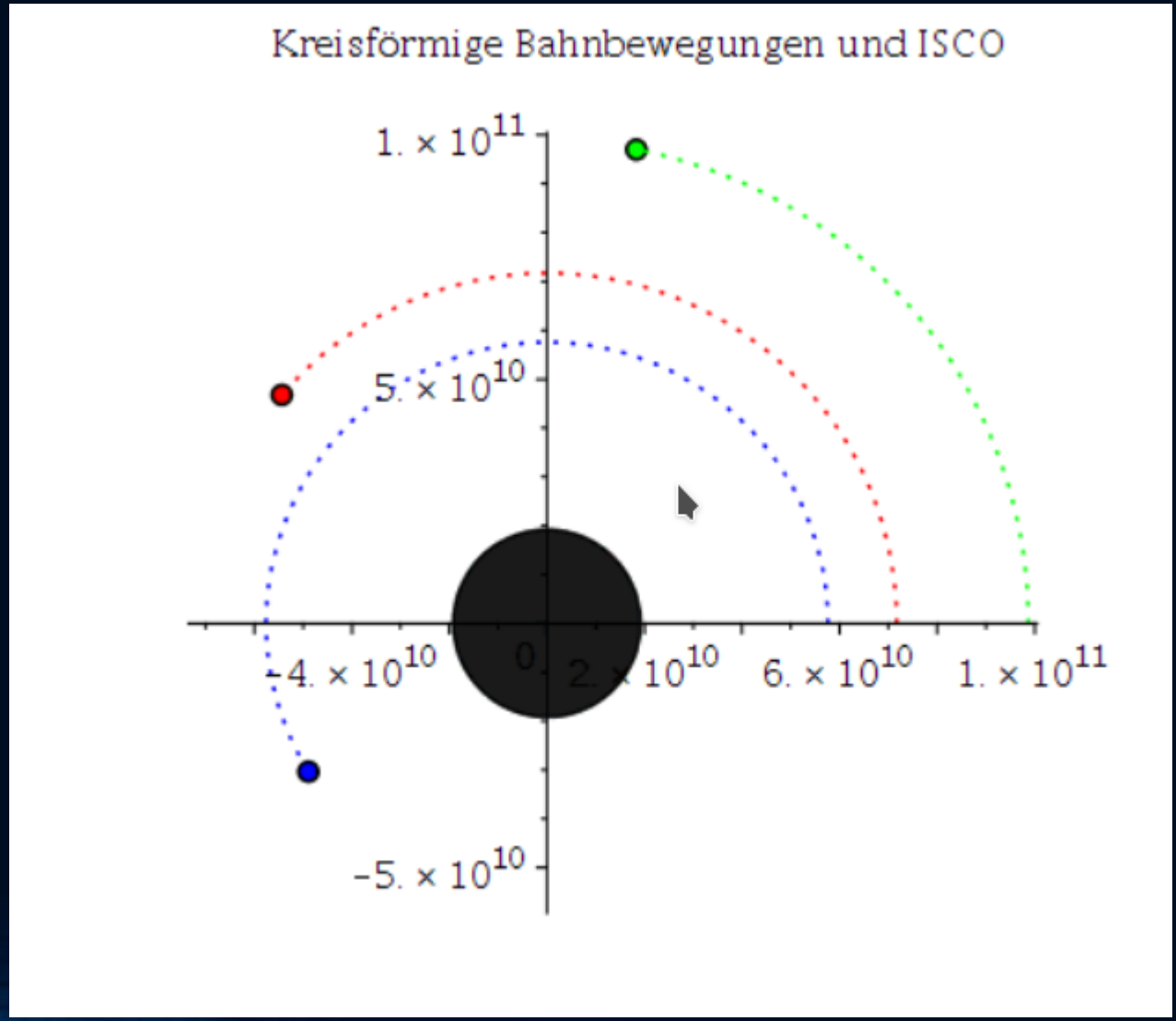
In der Literaturangabe 4 wird das Potential V(r,M,l) hingegen wie folgt definiert:

$$\text{2. Gleichung: } \rightarrow \left(\frac{dr}{d\tau}\right)^2 + (V(r, M, l))^2 = e^2$$

$$\text{wobei: } \quad V(r, M, l) = \sqrt{\left(1 - \frac{2M}{r}\right)\left(1 + \frac{l^2}{r^2}\right)} \quad .$$

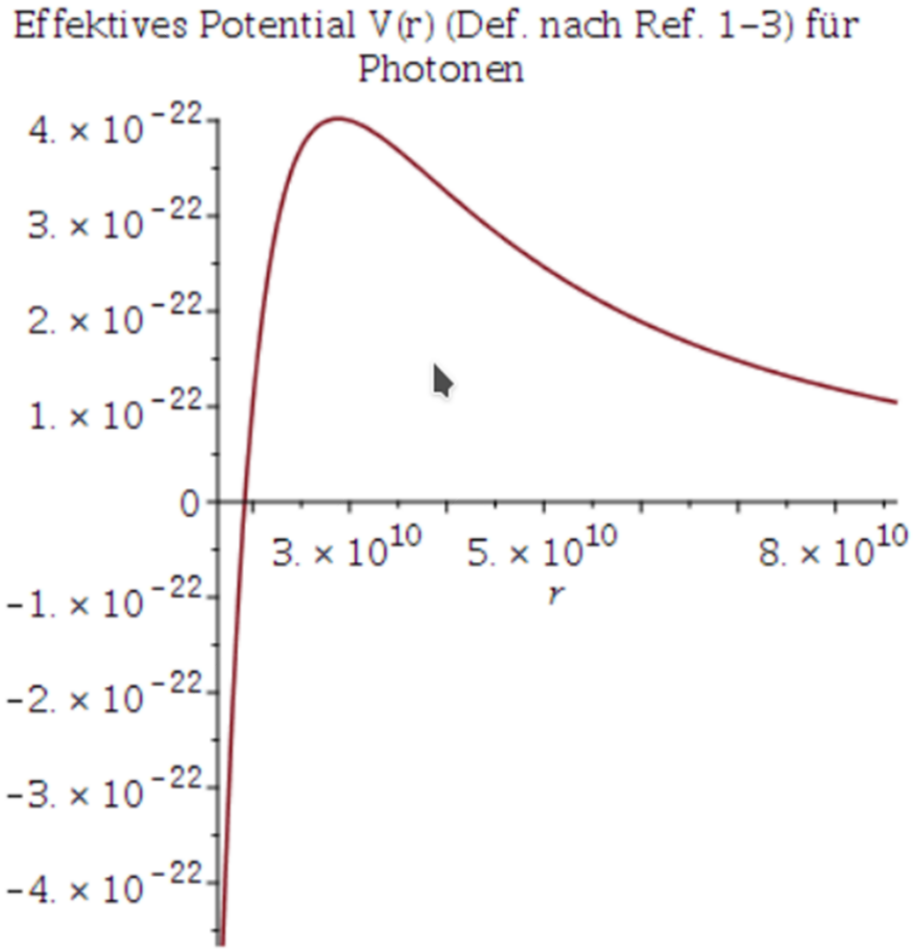
Das effektive Potential eines Probekörpers am ISCO hat eine Sattelpunktseigenschaft

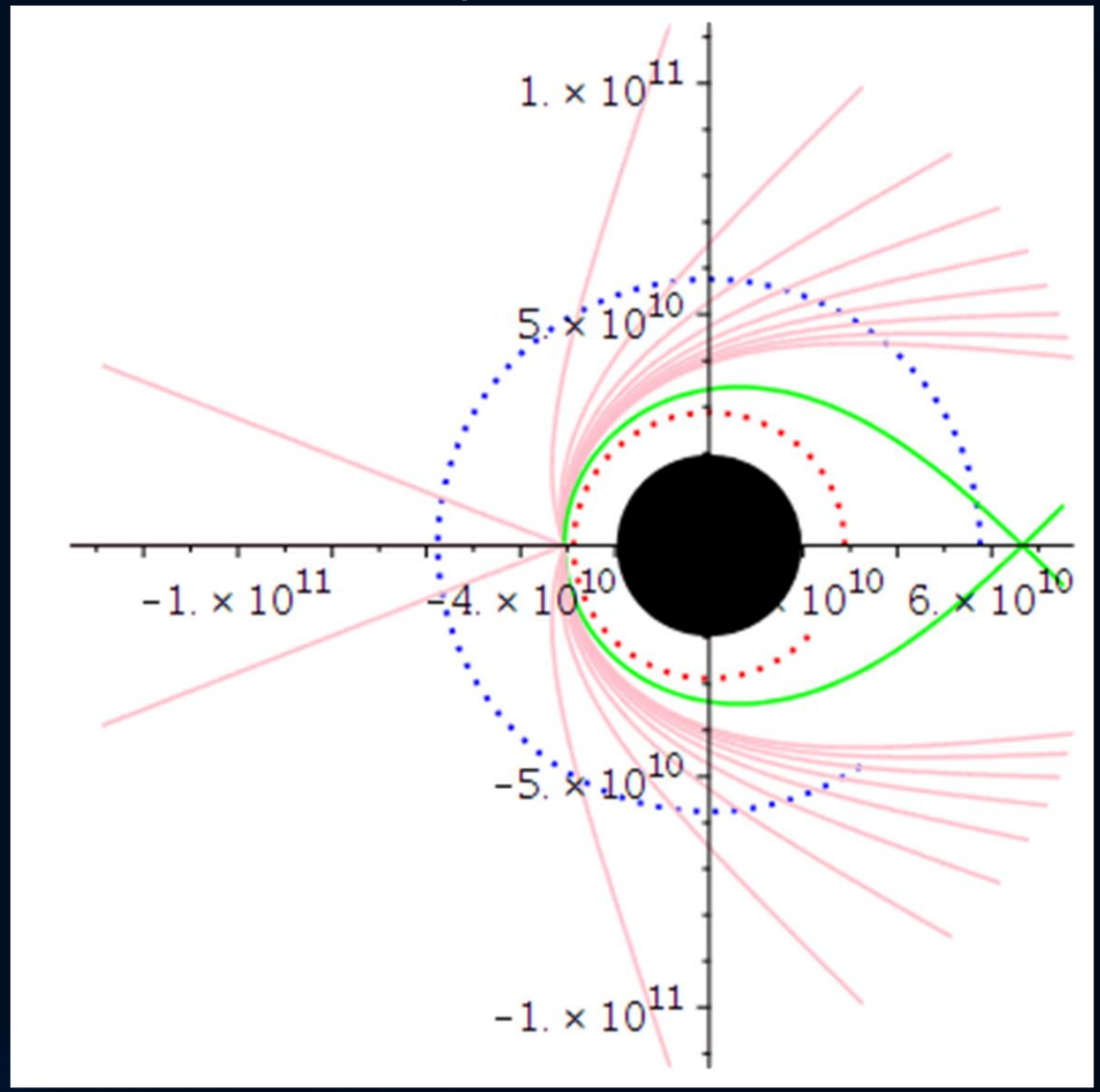




Masselose Teilchen (Photonen): Das effektive Potential und die Photonensphäre bei 3M

$$VeffPhotonHobson := (r, M) \mapsto \frac{1 - \frac{2M}{r}}{r^2}$$



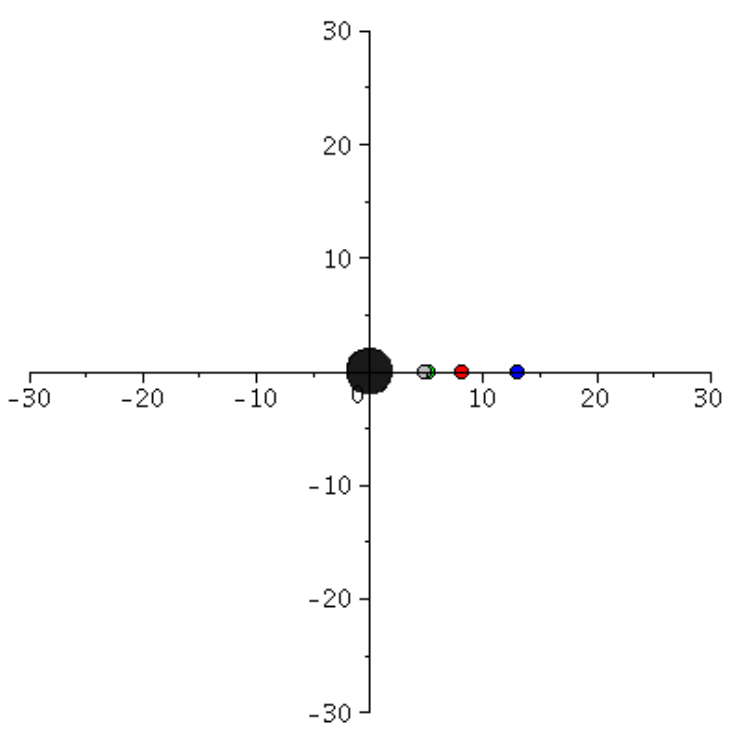


Blau: ISCO , Rot: Photonensphäre

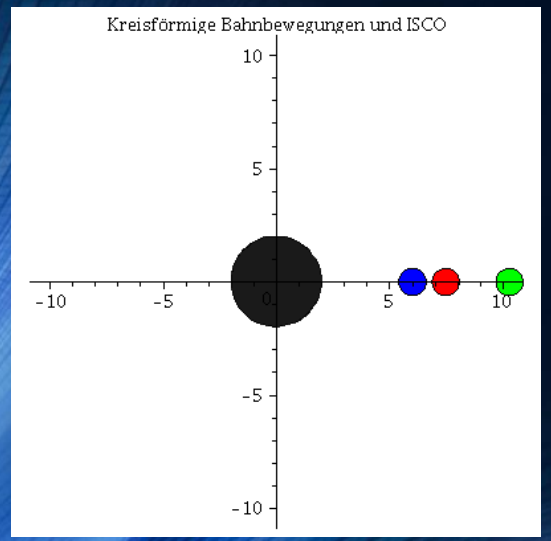
Geodätische Bewegung um ein nicht-rotierendes schwarzes Loch

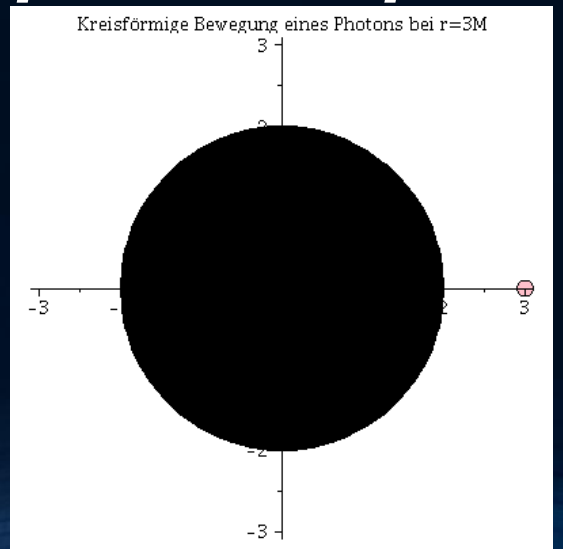
$$R_{\mu\nu} - \frac{1}{2} g_{\mu\nu} R = -8\pi T_{\mu\nu}$$

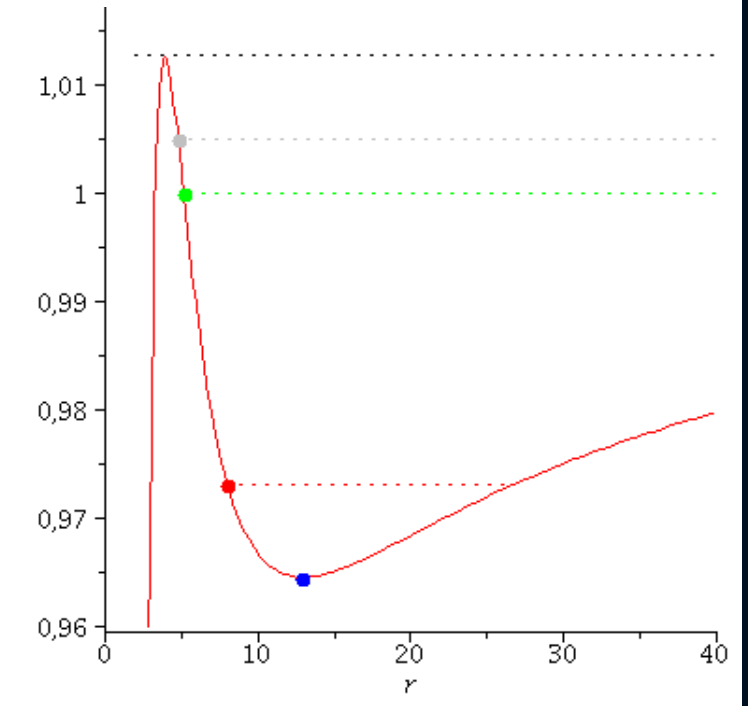
$$\frac{d^2 x^\mu}{d\tau^2} + \Gamma^\mu_{\nu\rho} \frac{dx^\nu}{d\tau} \frac{dx^\rho}{d\tau} = 0$$



The *ISCO* and the ***photon sphere***

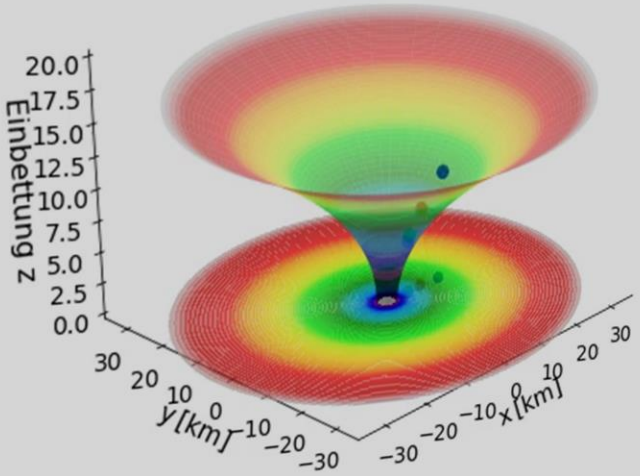


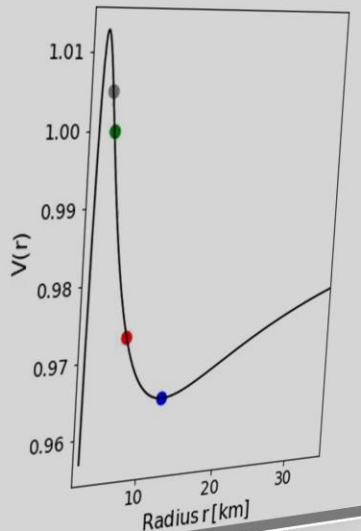




Jupyter Notebook

Klassifizierung unterschiedlicher Bahnbewegungen um ein nicht-rotierendes Schwarzes Loch





Bewegung eines Probekörpers um ein nicht-rotierendes schwarzes Loch in der Ebene

Teil II: Klassifizierung unterschiedlicher Bahnbewegungen

Im Folgenden wird die Geodätengleichung in vorgegebener Schwarzschild Raumzeit betrachtet. Die Geodätengleichung beschreibt wie sich ein Probekörper (Masse Probekörper $m << M$ Masse schwarzes Loch) im Raum bewegt und sagt voraus, dass diese Bewegung sich stets entlang der kürzesten Kurve, in der durch die Metrik beschriebenen gekrümmten Raumzeit, vollzieht.

Sie lässt sich demnach durch folgendes Variationsprinzip herleiten:

$$\int_A^B ds = \int_A^B \sqrt{g_{\mu\nu} dx^\mu dx^\nu} = \int_A^B \sqrt{g_{\mu\nu} \frac{dx^\mu}{d\lambda} \frac{dx^\nu}{d\lambda}} d\lambda \rightarrow \text{Extremal}$$

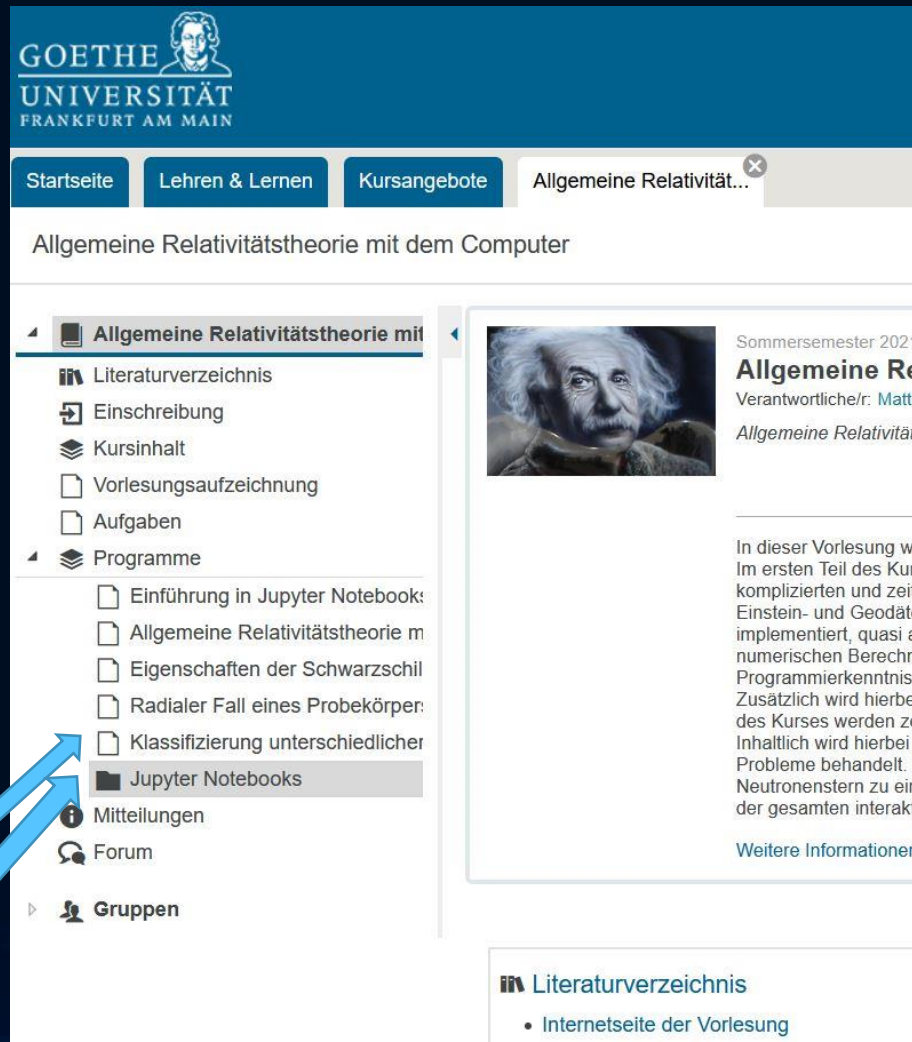
, wobei sich dann die Geodätengleichung mittels der Euler-Lagrange Gleichungen $L = \sqrt{g_{\mu\nu} \frac{dx^\mu}{d\lambda} \frac{dx^\nu}{d\lambda}}$, bzw. alternativ $L = g_{\mu\nu} \frac{dx^\mu}{d\lambda} \frac{dx^\nu}{d\lambda}$ ergibt:

$$\frac{d}{d\lambda}\left(\frac{\partial L}{\partial \dot{x}^\mu}\right) - \frac{\partial L}{\partial x^\mu} = 0 \quad \rightarrow \quad \frac{d^2 x^\mu}{d\lambda^2} + \Gamma^\mu_{\nu\rho} \frac{dx^\nu}{d\lambda} \frac{dx^\rho}{d\lambda} = 0$$

(λ z.B. die Eigenzeit τ des Probekörpers)

$\Gamma^\mu_{\nu\rho}$ sind die Christoffel Symbole zweiter Art

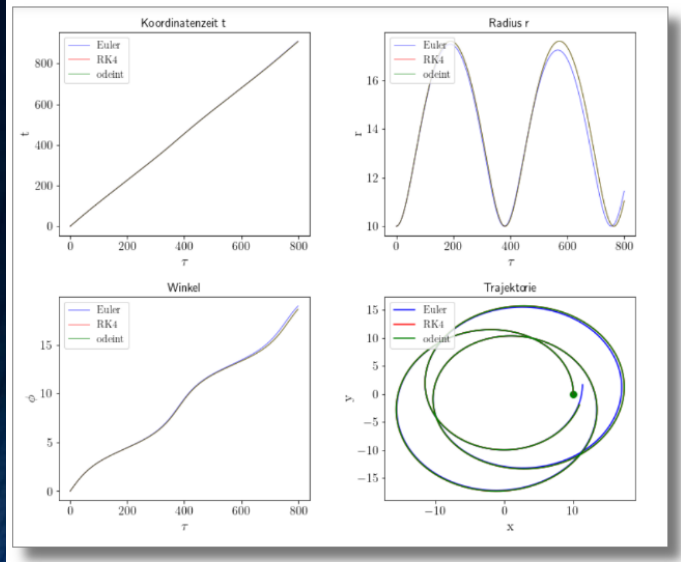
Auf der OLAT Seite des Kurses finden Sie die Jupyter Notebooks zum Ansehen und zum herunterladen



Vorlesung 4

Im ersten Abschnitt dieser Vorlesung lösen wir die Geodätengleichung mittels eines C/C++ Programms. Im zweiten Abschnitt dieser Vorlesung werden wir ein Jupyter Notebook behandeln, welches die Eigenschaften der innersten stabilen kreisförmigen Bahnbewegung eines massiven Probekörpers (ISCO, siehe rechtes Panel dieser Vorlesung) berechnet und diesen Grenzorbit verdeutlicht. Des Weiteren wird in diesem Notebook auch die Bewegung von masselosen Teilchen (z.B. Photonen) um ein nicht-rotierendes schwarzes Loch simuliert und die Photonensphäre eines schwarzen Loches berechnet. Die astrophysikalische Bedeutung des ISCOs und der Photonensphäre werden wir dann in der nächsten Vorlesung besprechen.

Numerisches Lösen der Geodätengleichung mittels eines C/C++ Programmes



Wir werden nun die Geodätengleichung mittels eines C++ Programmes lösen. Die Geodätengleichung ist ein System von vier gekoppelten Differentialgleichungen zweiter Ordnung, welches man in ein System von acht gekoppelten Differentialgleichungen erster Ordnung umschreiben und dann numerisch lösen kann. In diesem Unterpunkt beschränken wir uns auf eine Schwarzschild-Raumzeit (nicht-rotierendes, nicht-geladenes Schwarzes Loch). Aufgrund der sphärischen Symmetrie der Schwarzschild-Raumzeit kann man sich bei der Beschreibung auf ebene, äquatoriale Bewegungen beschränken ($\theta = \frac{\pi}{2}, \cos(\theta) = 0, \sin(\theta) = 1, \frac{d\theta}{d\tau} = 0, \frac{d^2\theta}{d\tau^2} = 0$). Dies hat zur Folge, dass eine der vier Differentialgleichungen trivial ist und man somit effektiv 'nur' sechs gekoppelte Differentialgleichungen erster Ordnung zu lösen hat. Wir leiten zunächst das Differentialgleichung-System der Geodätengleichung mittels eines Python Jupyter ab und exportieren es dann in ein C++ Programm. Wir benutzen dazu das Jupyter Notebook: GeodSchwarzschildII.ipynb und exportieren dann die Gleichungen des gekoppelten Differentialgleichung-Systems in C-Ausdrücke. Das C++ Programm können Sie sich unter folgendem Link herunterladen: DGL_2_VARTC.cpp.

Beim Klicken auf das nebenstehende linke Bild gelangen Sie zu dem Unterpunkt Numerisches Lösen der Geodätengleichung mittels eines C/C++ Programmes. Das Bild zeigt die Visualisierung der Daten von 'DGL_2_VARTC.dat' und vergleicht das Euler Verfahren mit der Runge-Kutta Ordnung vier Methode. Zusätzlich wird in der Abbildung auch die numerische Lösung, die in Python direkt generiert wurde (Methode "integrate.odeint()" im Python-Modul "scipy") mit den simulierten Daten der unterschiedlichen Verfahren verglichen (näheres siehe Numerisches Lösen der Geodätengleichung mittels eines C/C++ Programmes).

Die innerste stabile kreisförmige Bahnbewegung (der ISCO: Innermost Stable Circular Orbit) und die Photonensphäre

Die linke Seite der unteren Animation verdeutlicht die Trajektorien des ISCO und der Photonensphäre in einem eingebetteten Raumzeit-Diagramm der Schwarzschild-Metrik. Kreisförmige Bewegungen von massiven Probekörpern um ein nicht rotierendes schwarzes Loch können dem Loch nicht beliebig nahekommen. Die Grenze der Stabilität von kreisförmigen Trajektorien nennt man ISCO (siehe rechtes Panel dieser Vorlesung). Die linke untere Abbildung zeigt die simulierten Bahnen dreier kreisförmige Bewegungen, welche in unterschiedlichen Abstand um das schwarze Loch kreisen, wobei die Masse des schwarzen Loches auf

Vorlesung 4

In der vorigen Vorlesung hatten wir die Geodätengleichung in vorgegebener Schwarzschild-Raumzeit betrachtet und gezeigt, wie man eine Klassifizierung möglicher Bahnen von Probekörpern mittels eines definierten effektiven Potentials V(r,M,l) illustrieren kann (M ist die Masse des schwarzen Lochs und l der Bahndrehimpuls pro Masse m des Probekörpers). Eine dieser Bahnen ist von besonderer Bedeutung, die sogenannte innerste stabile kreisförmige Bahnbewegung (der ISCO: Innermost Stable Circular Orbit). Kreisförmige Bahnbewegungen sind dadurch charakterisiert, dass der Wert des Radiuses sich im laufe der Zeit nicht verändert und somit sich der radiale Abstand des Probekörpers vom schwarzen Loch gerade im Minimum des effektiven Potentials befindet. Es muss somit $\frac{dV}{dr} = 0$ gelten. Löst man diese Gleichung nach r auf, so erhält man zwei Lösungen, wobei die erste (positives Vorzeichen) dem stabilen Minimum und die zweite (negatives Vorzeichen) dem instabilen Maximum entspricht:

$$\frac{dV}{dr} = 0 \quad \Rightarrow r_{\pm} = \frac{l}{2M}\left(l \pm \sqrt{l^2 - 12M^2}\right)$$

Der ISCO hat gerade die Sattelpunkt-Eigenschaft, sodass zusätzlich $\frac{d^2V}{dr^2} = 0$ gelten muss. Der Drehimpuls l des Probekörpers und sein radialer Abstand vom schwarzen Loch nehmen die folgenden Werte an:

$$\frac{dV}{dr} = 0\ , \quad \frac{d^2V}{dr^2} = 0 \quad \underbrace{\Rightarrow}_{\text{ISCO}} \quad r = 6M\ , \quad l = 2\sqrt{3}M$$

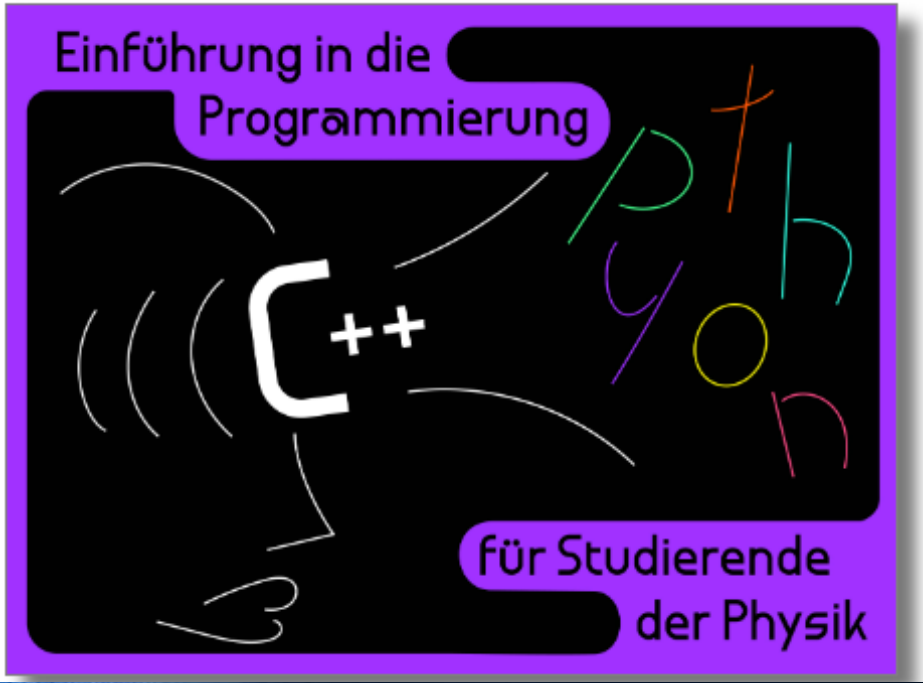
Kreisförmige Bewegungen um ein nicht-rotierendes schwarzes Loch sind somit nur bis zu einem Abstand von $r_{ISCO} = 6M$ möglich. Kommt man dem schwarzen Loch näher, endet man zwangsläufig in der echten Singularität. Dies gilt jedoch nur für massive Probekörper und nicht für masselose Teilchen (z.B. Photonen). Photonenbahnen unterscheiden im Wert des infinitesimalen Weglängenelementes $ds^2 = g_{\mu\nu}\, dx^{\mu} dx^{\nu}$. Mittels der Eigenschaft für Photonen $\left(\frac{ds}{d\tau}\right)^2 = 0$ vereinfacht sich die radiale Gleichung der Geodätengleichung und man kann ein effektives Potential definieren, welches allein von der Masse des schwarzen Lochs abhängt:

$$\frac{1}{l^2}\left(\frac{dr}{d\lambda}\right)^2 + V_{Photon}(r) = \frac{1}{b^2}$$

$$V_{Photon}(r) = \frac{1}{r^2}\left(1 - \frac{2M}{r}\right) \quad ,$$

wobei der Parameter b der Impaktparameter der Photonenbahn darstellt

C++: Systeme von gekoppelten Differentialgleichungen und Differentialgleichungen zweiter Ordnung



Im vorigen Unterpunkt hatten wir die Geodätengleichung (bzw. das im rechten Panel dieser Vorlesung dargestellte System von gekoppelten Differentialgleichungen zweiter Ordnung) mittels der im Python Modul 'scipy' definierter Funktion "odeint(DGLsys, initialval, tauval)" numerisch gelöst. In diesem neuen Unterpunkt werden wir die Grundlagen der Programmiersprache C++ wiederholen und das numerische Lösen von Differentialgleichungen kurz erläutern (das Material basiert auf der Vorlesungsreihe Einführung in die Programmierung für Studierende der Physik (SS 2022). In der nächsten Vorlesung werden wir dann die Geodätengleichung mittels eines C++ Programmes lösen. Das numerische Lösen von Differentialgleichungen ist ein mathematisch anspruchsvolles Thema und kann in dieser Vorlesung nicht im Detail erläutert werden. Die Geodätengleichung ist ein System von vier gekoppelten Differentialgleichungen zweiter Ordnung, welches man in ein System von acht gekoppelten Differentialgleichungen erster Ordnung umschreiben kann. In diesem Unterpunkt beschreiben wir die Vorgehensweise, wie man solche Differentialgleichungen höherer Ordnung numerisch mittels eines C++ Programmes löst. In diesem Unterpunkt werden wir uns zunächst mit Systemen von gekoppelten Differentialgleichungen erster Ordnung befassen und dann das numerische Lösen von Differentialgleichungen zweiter Ordnung vorstellen (näheres siehe Systeme von gekoppelten Differentialgleichungen und Differentialgleichungen zweiter Ordnung).

Numerisches Lösen der Geodätengleichung mittels eines C/C++ Programmes

In Vorlesung 3 hatten wir die Geodätengleichung (bzw. das im unteren Bild dargestellte System von gekoppelten Differentialgleichungen zweiter Ordnung) mittels eines Python Jupyter Notebooks hergeleitet und danach unter Zuhilfenahme der im Python Modul 'scipy' definierten Funktion "odeint(DGLsys, initialval, tauval)" numerisch gelöst. In diesem neuen Unterpunkt werden wir die Geodätengleichung mittels eines C/C++ Programmes lösen (die Vorgehensweise dabei basiert auf dem in der Vorlesungsreihe Einführung in die Programmierung für Studierende der Physik (SS 2022) vorgestellten C++-Programm DGL_2.cpp. Das numerische Lösen von Differentialgleichungen ist ein mathematisch anspruchsvolles Thema und kann in dieser Vorlesung nicht im Detail erläutert werden. Bitte sehen Sie sich, bevor Sie weiterlesen, die folgende Seite an: Systeme von gekoppelten Differentialgleichungen und Differentialgleichungen zweiter Ordnung (SS 2022)

Die Geodätengleichung ist ein System von vier gekoppelten Differentialgleichungen zweiter Ordnung, welches man in ein System von acht gekoppelten Differentialgleichungen erster Ordnung umschreiben und dann numerisch lösen kann. In diesem Unterpunkt beschränken wir uns auf eine Schwarzschild-Raumzeit (nicht-rotierendes, nicht-geladenes Schwarzes Loch). Aufgrund der sphärischen Symmetrie der Schwarzschild-Raumzeit kann man sich bei der Beschreibung auf ebene, äquatoriale Bewegungen beschränken ($\theta = \frac{\pi}{2}, \cos(\theta) = 0, \sin(\theta) = 1, \frac{d\theta}{d\tau} = 0, \frac{d^2\theta}{d\tau^2} = 0$). Dies hat zur Folge, dass eine der vier Differentialgleichungen trivial ist und man somit effektiv 'nur' sechs gekoppelte Differentialgleichungen erster Ordnung zu lösen hat. Im dem folgenden Unterpunkt leiten wir zunächst das Differentialgleichung-System der Geodätengleichung mittels eines Python Jupyter ab und exportieren es dann in ein C++ Programm. Das C++ Programm, wird dann im darauf folgenden Unterpunkt erläutert. Abschließend werden die Ergebnisse vorgestellt und diskutiert.

Analytische Herleitung des DGL Systems mittels Python Jupyter

Zunächst müssen wir das Differentialgleichung-System der Geodätengleichung in eine für das C++-Programm verständliche Form bringen. Wir benutzen dazu das Jupyter Notebook: GeodSchwarzschildII.ipynb und exportieren dann die Gleichungen des gekoppelten Differentialgleichung-Systems in C-Ausdrücke. Wie schon erwähnt betrachten wir eine ebene Bewegung des Probekörpers um ein Schwarzes Loch, wobei nur drei der vier Geodätengleichungen relevant sind (siehe linkes unteres Bild):

Beispiel einer gebundenen elliptischen Bahn

Wir lassen im Folgenden nur ebene Bewegungen zu ($\theta = \frac{\pi}{2}, \cos(\theta) = 0, \sin(\theta) = 1, \frac{d\theta}{d\tau} = 0, \frac{d^2\theta}{d\tau^2} = 0$). Unter diesen Annahmen sind nur drei der vier Geodätengleichungen relevant und diese lauten dann:

In [7]:
```
GeodEq0=GeodEq0.subs({(theta,pi/2),(diff(x2,tau),0)})
GeodEq1=GeodEq1.subs({(theta,pi/2),(diff(x2,tau),0)})
GeodEq3=GeodEq3.subs({(theta,pi/2),(diff(x2,tau),0)})

GeodEq0_a = Eq(diff(x0,tau,tau),solve(GeodEq0,diff(x0,tau,tau))[0])
GeodEq1_a = Eq(diff(x1,tau,tau),solve(GeodEq1,diff(x1,tau,tau))[0])
GeodEq3_a = Eq(diff(x3,tau,tau),solve(GeodEq3,diff(x3,tau,tau))[0])
```

In [8]:
```
GeodEq0_a
```

Out[8]:
$$\frac{d^2}{d\tau^2}t(\tau) = \frac{2M\frac{d}{d\tau}r(\tau)\frac{d}{d\tau}t(\tau)}{r(2M-r)}$$

In [9]:
```
GeodEq1_a
```

Out[9]:
$$\frac{d^2}{d\tau^2}r(\tau) = -\frac{2M^2\left(\frac{d}{d\tau}r(\tau)\right)^2}{r(2M-r)^2} + \frac{2M^2\left(\frac{d}{d\tau}t(\tau)\right)^2}{r^3} + \frac{M\left(\frac{d}{d\tau}r(\tau)\right)^2}{(2M-r)^2} - \frac{M\left(\frac{d}{d\tau}t(\tau)\right)^2}{r^2} + \frac{(-2M+r)^3\left(\frac{d}{d\tau}\phi(\tau)\right)^2}{(2M-r)^2}$$

In [10]:
```
GeodEq3_a
```

Out[10]:
$$\frac{d^2}{d\tau^2}\phi(\tau) = -\frac{2\frac{d}{d\tau}\phi(\tau)\frac{d}{d\tau}r(\tau)}{r}$$

$$t(\tau) = y_1(\tau), \quad r(\tau) = y_2(\tau), \quad \phi(\tau) = y_3(\tau), \quad \frac{dt(\tau)}{d\tau} = y_4(\tau), \quad \frac{dr(\tau)}{d\tau} = y_5(\tau), \quad \frac{d\phi(\tau)}{d\tau} = y_6$$

und schreiben unser DGL-System wie folgt um:

In [11]:
```
y1, y2, y3, y4, y5, y6 = symbols('y_1, y_2, y_3, y_4, y_5, y_6')
```

In [12]:
```
SysDGL_0 = Eq(diff(x0,tau,tau),GeodEq0_a.rhs.subs({(r,y2),(diff(x0,tau),y4),(diff(x1,tau),y5)}))
SysDGL_0
```

Out[12]:
$$\frac{d^2}{d\tau^2}t(\tau) = \frac{2My_4y_5}{y_2(2M-y_2)}$$

In [13]:
```
SysDGL_1 = Eq(diff(x1,tau,tau),GeodEq1_a.rhs.subs({(r,y2),(diff(x0,tau),y4),(diff(x1,tau),y5),(diff(x3,tau),y6)}))
SysDGL_1
```

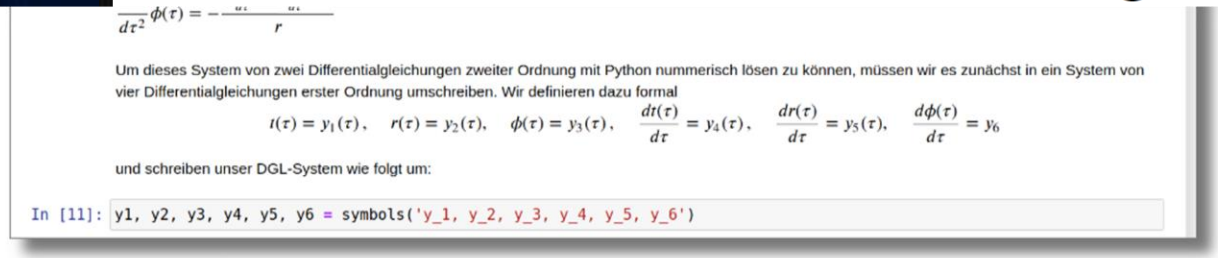
Out[13]:
$$\frac{d^2}{d\tau^2}r(\tau) = -\frac{2M^2y_5^2}{y_2(2M-y_2)^2} + \frac{2M^2y_4^2}{y_2^3} + \frac{My_5^2}{(2M-y_2)^2} - \frac{My_4^2}{y_2^2} + \frac{y_6^2(-2M+y_2)^3}{(2M-y_2)^2}$$

In [14]:
```
SysDGL_3 = Eq(diff(x1,tau,tau),GeodEq3_a.rhs.subs({(r,y2),(diff(x0,tau),y4),(diff(x1,tau),y5),(diff(x3,tau),y6)}))
SysDGL_3
```

Out[14]:
$$\frac{d^2}{d\tau^2}r(\tau) = -\frac{2y_5y_6}{y_2}$$

Dieses System von drei gekoppelten DGLs 2. Ordnung in der Eigenzeit τ werden wir später zusätzlich mittels der Programmiersprache C++ lösen. Wir exportieren deshalb schon jetzt das zugrundeliegende System von DGLs als C++ Code:

Numerisches Lösen der Geodätengleichung mittels eines C/C++ Programmes



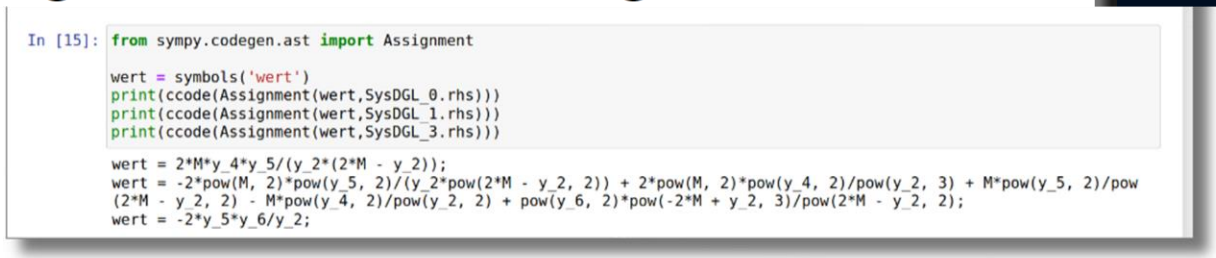
$$\frac{d^2}{d\tau^2}\phi(\tau) = -\frac{\quad}{r}$$

Um dieses System von zwei Differentialgleichungen zweiter Ordnung mit Python nummerisch lösen zu können, müssen wir es zunächst in ein System von vier Differentialgleichungen erster Ordnung umschreiben. Wir definieren dazu formal

$$t(\tau) = y_1(\tau)\,, \quad r(\tau) = y_2(\tau), \quad \phi(\tau) = y_3(\tau)\,, \quad \frac{dt(\tau)}{d\tau} = y_4(\tau)\,, \quad \frac{dr(\tau)}{d\tau} = y_5(\tau), \quad \frac{d\phi(\tau)}{d\tau} = y_6$$

und schreiben unser DGL-System wie folgt um:

```
In [11]: y1, y2, y3, y4, y5, y6 = symbols('y_1, y_2, y_3, y_4, y_5, y_6')
```



```
In [15]: from sympy.codegen.ast import Assignment

         wert = symbols('wert')
         print(ccode(Assignment(wert,SysDGL_0.rhs)))
         print(ccode(Assignment(wert,SysDGL_1.rhs)))
         print(ccode(Assignment(wert,SysDGL_3.rhs)))
```

```
wert = 2*M*y_4*y_5/(y_2*(2*M - y_2));
wert = -2*pow(M, 2)*pow(y_5, 2)/(y_2*pow(2*M - y_2, 2)) + 2*pow(M, 2)*pow(y_4, 2)/pow(y_2, 3) + M*pow(y_5, 2)/pow
(2*M - y_2, 2) - M*pow(y_4, 2)/pow(y_2, 2) + pow(y_6, 2)*pow(-2*M + y_2, 3)/pow(2*M - y_2, 2);
wert = -2*y_5*y_6/y_2;
```

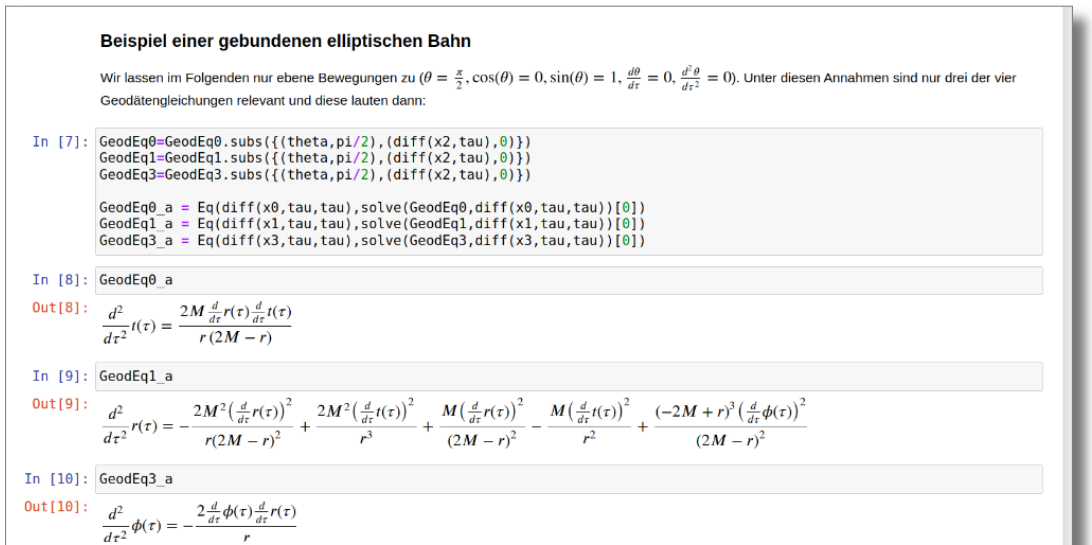
Um dieses System von drei gekoppelten Differentialgleichungen zweiter Ordnung mit Python nummerisch lösen zu können, müssen wir es zunächst in ein System von sechs Differentialgleichungen erster Ordnung umschreiben. Wir definieren dazu formal

$$t(\tau) = y_1(\tau)\,, \quad r(\tau) = y_2(\tau), \quad \phi(\tau) = y_3(\tau)\,, \quad \frac{dt(\tau)}{d\tau} = y_4(\tau)\,, \quad \frac{dr(\tau)}{d\tau} = y_5(\tau), \quad \frac{d\phi(\tau)}{d\tau} = y_6$$

und schreiben unser DGL-System um (siehe rechtes oberes Bild).

Analytische Herleitung des DGL Systems mittels Python Jupyter

Zunächst müssen wir das Differentialgleichung-System der Geodätengleichung in eine für das C++-Programm verständliche Form bringen. Wir benutzen dazu das Jupyter Notebook: GeodSchwarzschildII.ipynb und exportieren dann die Gleichungen des gekoppelten Differentialgleichung-Systems in C-Ausdrücke. Wie schon erwähnt betrachten wir eine ebene Bewegung des Probekörpers um ein Schwarzes Loch, wobei nur drei der vier Geodätengleichungen relevant sind (siehe linkes unteres Bild):



Beispiel einer gebundenen elliptischen Bahn

Wir lassen im Folgenden nur ebene Bewegungen zu ($\theta = \frac{\pi}{2}, \cos(\theta) = 0, \sin(\theta) = 1, \frac{d\theta}{d\tau} = 0, \frac{d^2\theta}{d\tau^2} = 0$). Unter diesen Annahmen sind nur drei der vier Geodätengleichungen relevant und diese lauten dann:

```
In [7]: GeodEq0=GeodEq0.subs({(theta,pi/2),(diff(x2,tau),0)})
        GeodEq1=GeodEq1.subs({(theta,pi/2),(diff(x2,tau),0)})
        GeodEq3=GeodEq3.subs({(theta,pi/2),(diff(x2,tau),0)})

        GeodEq0_a = Eq(diff(x0,tau,tau),solve(GeodEq0,diff(x0,tau,tau))[0])
        GeodEq1_a = Eq(diff(x1,tau,tau),solve(GeodEq1,diff(x1,tau,tau))[0])
        GeodEq3_a = Eq(diff(x3,tau,tau),solve(GeodEq3,diff(x3,tau,tau))[0])
```

```
In [8]: GeodEq0_a
```

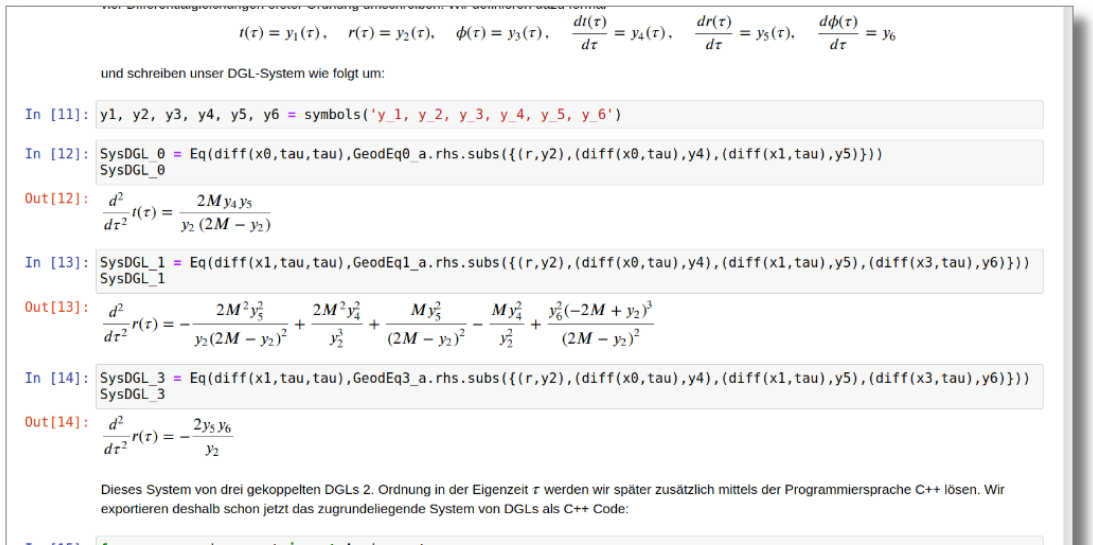
Out[8]:
$$\frac{d^2}{d\tau^2}t(\tau) = \frac{2M\frac{d}{d\tau}r(\tau)\frac{d}{d\tau}t(\tau)}{r(2M-r)}$$

```
In [9]: GeodEq1_a
```

Out[9]:
$$\frac{d^2}{d\tau^2}r(\tau) = -\frac{2M^2\left(\frac{d}{d\tau}r(\tau)\right)^2}{r(2M-r)^2} + \frac{2M^2\left(\frac{d}{d\tau}t(\tau)\right)^2}{r^3} + \frac{M\left(\frac{d}{d\tau}r(\tau)\right)^2}{(2M-r)^2} - \frac{M\left(\frac{d}{d\tau}t(\tau)\right)^2}{r^2} + \frac{(-2M+r)^3\left(\frac{d}{d\tau}\phi(\tau)\right)^2}{(2M-r)^2}$$

```
In [10]: GeodEq3_a
```

Out[10]:
$$\frac{d^2}{d\tau^2}\phi(\tau) = -\frac{2\frac{d}{d\tau}\phi(\tau)\frac{d}{d\tau}r(\tau)}{r}$$



$$t(\tau) = y_1(\tau)\,, \quad r(\tau) = y_2(\tau), \quad \phi(\tau) = y_3(\tau)\,, \quad \frac{dt(\tau)}{d\tau} = y_4(\tau)\,, \quad \frac{dr(\tau)}{d\tau} = y_5(\tau), \quad \frac{d\phi(\tau)}{d\tau} = y_6$$

und schreiben unser DGL-System wie folgt um:

```
In [11]: y1, y2, y3, y4, y5, y6 = symbols('y_1, y_2, y_3, y_4, y_5, y_6')
```

```
In [12]: SysDGL_0 = Eq(diff(x0,tau,tau),GeodEq0_a.rhs.subs({(r,y2),(diff(x0,tau),y4),(diff(x1,tau),y5)}))
         SysDGL_0
```

Out[12]:
$$\frac{d^2}{d\tau^2}t(\tau) = \frac{2My_4y_5}{y_2(2M-y_2)}$$

```
In [13]: SysDGL_1 = Eq(diff(x1,tau,tau),GeodEq1_a.rhs.subs({(r,y2),(diff(x0,tau),y4),(diff(x1,tau),y5),(diff(x3,tau),y6)}))
         SysDGL_1
```

Out[13]:
$$\frac{d^2}{d\tau^2}r(\tau) = -\frac{2M^2y_5^2}{y_2(2M-y_2)^2} + \frac{2M^2y_4^2}{y_2^3} + \frac{My_5^2}{(2M-y_2)^2} - \frac{My_4^2}{y_2^2} + \frac{y_6^2(-2M+y_2)^3}{(2M-y_2)^2}$$

```
In [14]: SysDGL_3 = Eq(diff(x1,tau,tau),GeodEq3_a.rhs.subs({(r,y2),(diff(x0,tau),y4),(diff(x1,tau),y5),(diff(x3,tau),y6)}))
         SysDGL_3
```

Out[14]:
$$\frac{d^2}{d\tau^2}r(\tau) = -\frac{2y_5y_6}{y_2}$$

Dieses System von drei gekoppelten DGLs 2. Ordnung in der Eigenzeit τ werden wir später zusätzlich mittels der Programmiersprache C++ lösen. Wir exportieren deshalb schon jetzt das zugrundeliegende System von DGLs als C++ Code:

$$\frac{d^2\phi(\tau)}{d\tau^2} = -\frac{\dots}{r}$$

Um dieses System von zwei Differentialgleichungen zweiter Ordnung mit Python nummerisch lösen zu können, müssen wir es zunächst in ein System von vier Differentialgleichungen erster Ordnung umschreiben. Wir definieren dazu formal

$$t(\tau) = y_1(\tau)\,, \quad r(\tau) = y_2(\tau), \quad \phi(\tau) = y_3(\tau)\,, \quad \frac{dt(\tau)}{d\tau} = y_4(\tau)\,, \quad \frac{dr(\tau)}{d\tau} = y_5(\tau), \quad \frac{d\phi(\tau)}{d\tau} = y_6$$

und schreiben unser DGL-System wie folgt um:

In [11]:
```
y1, y2, y3, y4, y5, y6 = symbols('y_1, y_2, y_3, y_4, y_5, y_6')
```

In [15]:
```
from sympy.codegen.ast import Assignment

wert = symbols('wert')
print(ccode(Assignment(wert,SysDGL_0.rhs)))
print(ccode(Assignment(wert,SysDGL_1.rhs)))
print(ccode(Assignment(wert,SysDGL_3.rhs)))
```

```
wert = 2*M*y_4*y_5/(y_2*(2*M - y_2));
wert = -2*pow(M, 2)*pow(y_5, 2)/(y_2*pow(2*M - y_2, 2)) + 2*pow(M, 2)*pow(y_4, 2)/pow(y_2, 3) + M*pow(y_5, 2)/pow
(2*M - y_2, 2) - M*pow(y_4, 2)/pow(y_2, 2) + pow(y_6, 2)*pow(-2*M + y_2, 3)/pow(2*M - y_2, 2);
wert = -2*y_5*y_6/y_2;
```

Um dieses System von drei gekoppelten Differentialgleichungen zweiter Ordnung mit P… nummerisch lösen zu können, müssen wir es zunächst in ein System von sechs Differentialgleichungen erster Ordnung …reiben. Wir definieren dazu formal

$$t(\tau) = y_1(\tau)\,, \quad r(\tau) = y_2(\tau), \quad \phi(\tau) = y_3(\tau\dots \quad \frac{\dots(\tau)}{d\tau} = y_4(\tau)\,, \quad \frac{dr(\tau)}{d\tau} = y_5(\tau), \quad \frac{d\phi(\tau)}{d\tau} = y_6$$

und schreiben unser DG…em um (siehe rechtes oberes Bild).

```
#include <cmath>              // Bibliothek für mathematisches (e-Funktion, Betrag, ...)

double f_1(double y_1, double y_2, double y_3, double y_4, double y_5, double y_6){ // Dek… und Definition der Funktion f_1
    double wert;
    wert = y_4;                                                                     // E…he Definition der Funktion
    return wert;                                                                    // …ewert der Funktion f_1
}                                                                                   // …r Funktion f_1

double f_2(double y_1, double y_2, double y_3, double y_4, double y_5, double y_6){ …ration und Definition der Funktion f_2
    double wert;
    wert = y_5;                                                                     …gentliche Definition der Funktion
    return wert;                                                                    …eckgabewert der Funktion f_2
}                                                                                   …nde der Funktion f_2

double f_3(double y_1, double y_2, double y_3, double y_4, double y_5, double …     Deklaration und Definition der Funktion f_3 (DGL fuer ddt/ddtau)
    double M = 1;                                                                   // gravitativ wirkende Masse M = 1
    double wert;
    wert = y_6;                                                                     // Eigentliche Definition der Funktion
    return wert;                                                                    // Rueckgabewert der Funktion f_3
}                                                                                   // Ende der Funktion f_3

double f_4(double y_1, double y_2, double y_3, double y_4, double y_5…_6){ // Deklaration und Definition der Funktion f_4 (DGL fuer ddr/ddtau)
    double M = 1;                                                                   // gravitativ wirkende Masse M = 1
    double wert;
    wert = 2*M*y_4*y_5/(y_2*(2*M - y_2));                                           // Eigentliche Definition der Funktion
    return wert;                                                                    // Rueckgabewert der Funktion f_4
}                                                                                   // Ende der Funktion f_4

double f_5(double y_1, double y_2, double y_3, double y_4, double y_5, double y_6){ // Deklaration und Definition der Funktion f_5 (DGL fuer ddphi/ddtau)
    double M = 1;                                                                   // gravitativ wirkende Masse M = 1
    double wert;
    // Eigentliche Definition der Funktion
    wert = -2*pow(M, 2)*pow(y_5, 2)/(y_2*pow(2*M - y_2, 2)) + 2*pow(M, 2)*pow(y_4, 2)/pow(y_2, 3) + M*pow(y_5, 2)/pow(2*M - y_2, 2) - M*pow(y_4, 2)/pow(y_2, 2) + pow(y_6, 2)*pow(-2*M + y_2, 3)/pow(2*M - y_2, 2);
    return wert;                                                                    // Rueckgabewert der Funktion f_5
}                                                                                   // Ende der Funktion f_5

double f_6(double y_1, double y_2, double y_3, double y_4, double y_5, double y_6){ // Deklaration und Definition der Funktion f_6
    double wert;
    wert = -2*y_5*y_6/y_2;                                                          // Eigentliche Definition der Funktion
    return wert;                                                                    // Rueckgabewert der Funktion f_6
}                                                                                   // Ende der Funktion f_6

int main(){                                                                         // Hauptfunktion
    double tau;                                                                     // Aktueller Eigenzeit-Wert
    double a = 0;                                                                   // Untergrenze des Eigenzeit-Intervalls [a,b] in dem die Loesung berechnet werden soll
    double b = 800;                                                                 // Obergrenze des Intervalls [a,b]
```

In der main()-Funktion des Programmes werden am Anfang die nötigen Variablen definiert, die man für die Euler bzw. Runge-Kutta Ordnung vier Methode benötigt. Die nötigen sechs Anfangswerte $\alpha_i(\tau = 0)\,,\ i = 1, 2, ..6$ werden dem Beispiel des Jupyter Notebooks GeodSchwarzschildII.ipynb entnommen. Die Simulation der Bewegung des Probekörpers erfolgt in einem Eigenzeit-Intervall $[a, b] = [0, 800]$, wobei wir $N = 10000$ Gitterpunkte verwenden.

Die oben ausgegebenen Ausdrücke werden nun in das C++-Programm integriert.

Das C++ Programm zum Lösen der Geodätengleichung

Als Vorlage verwenden wir das C++-Programm DGL_2.cpp und implementieren zunächst die ausgegebenen Ausdrücke in sechs DGL-bestimmenden C++-Funktionen

Öffnen

DGL_2_VARTC.cpp
~/neu_2022/VARTC/www/VARTC_2023

```cpp
/* VARTC 2023 (siehe http://itp.uni-frankfurt.de/~hanauske/VARTC/VARTC2023.html , Vorlesung 4)
 * basierend auf VPROG 2022 (siehe https://itp.uni-frankfurt.de/~hanauske/VPROG/index.html , Vorlesung 9)
 * C/C++ Programm zum Loesen der Geodaetengleichung
 * in vorgegebener Schwarzschild-Raumzeit (nicht-rotierendes, nicht-geladenes Schwarzes Loch)
 * Die Loesung benutzt die Euler bzw. Runge-Kutta Ordnung vier Methode
 * Koordinaten (Koordinatenzeit: t=y_1 , Radius: r=y_2 , Winkel: phi=y_3, Winkel: theta=pi/2=const)
 * Die zeitliche Entwicklung erfolgt in Eigenzeit tau von y_1(tau), y_2(tau), y_3(tau)
 *  Ausgabe zum Plotten mittels Python Jupyter Notebook DGL_2_VARTC.ipynb: "./a.out > DGL_2_VARTC.dat
 */

#include <stdio.h>            // Standard Input- und Output Bibliothek in C, z.B. printf(...)
#include <cmath>              // Bibliothek für mathematisches (e-Funktion, Betrag, ...)

double f_1(double y_1, double y_2, double y_3, double y_4, double y_5, double y_6){ // Deklaration und Definition der Funktion f_1
    double wert;
    wert = y_4;                                                                     // Eigentliche Definition der Funktion
    return wert;                                                                    // Rueckgabewert der Funktion f_1
}                                                                                   // Ende der Funktion f_1

double f_2(double y_1, double y_2, double y_3, double y_4, double y_5, double y_6){ // Deklaration und Definition der Funktion f_2
    double wert;
    wert = y_5;                                                                     // Eigentliche Definition der Funktion
    return wert;                                                                    // Rueckgabewert der Funktion f_2
}                                                                                   // Ende der Funktion f_2

double f_3(double y_1, double y_2, double y_3, double y_4, double y_5, double y_6){ // Deklaration und Definition der Funktion f_3 (DGL fuer dt/ddtau)
    double M = 1;                                                                   // gravitativ wirkende Masse M = 1
    double wert;
    wert = y_6;                                                                     // Eigentliche Definition der Funktion
    return wert;                                                                    // Rueckgabewert der Funktion f_3
}                                                                                   // Ende der Funktion f_3

double f_4(double y_1, double y_2, double y_3, double y_4, double y_5, double y_6){ // Deklaration und Definition der Funktion f_4 (DGL fuer dr/ddtau)
    double M = 1;                                                                   // gravitativ wirkende Masse M = 1
    double wert;
    wert = 2*M*y_4*y_5/(y_2*(2*M - y_2));                                           // Eigentliche Definition der Funktion
    return wert;                                                                    // Rueckgabewert der Funktion f_4
}                                                                                   // Ende der Funktion f_4

double f_5(double y_1, double y_2, double y_3, double y_4, double y_5, double y_6){ // Deklaration und Definition der Funktion f_5 (DGL fuer ddphi/ddtau)
    double M = 1;                                                                   // gravitativ wirkende Masse M = 1
    double wert;
    // Eigentliche Definition der Funktion
    wert = -2*pow(M, 2)*pow(y_5, 2)/(y_2*pow(2*M - y_2, 2)) + 2*pow(M, 2)*pow(y_4, 2)/pow(y_2, 3) + M*pow(y_5, 2)/pow(2*M - y_2, 2) - M*pow(y_4, 2)/pow(y_2, 2) + pow(y_6, 2)*pow(-2*M + y_2, 3)/pow(2*M - y_2, 2);
    return wert;                                                                    // Rueckgabewert der Funktion f_5
}                                                                                   // Ende der Funktion f_5

double f_6(double y_1, double y_2, double y_3, double y_4, double y_5, double y_6){ // Deklaration und Definition der Funktion f_6
    double wert;
    wert = -2*y_5*y_6/y_2;                                                          // Eigentliche Definition der Funktion
    return wert;                                                                    // Rueckgabewert der Funktion f_6
}                                                                                   // Ende der Funktion f_6

int main(){                                                                         // Hauptfunktion
    double tau;                                                                     // Aktueller Eigenzeit-Wert
    double a = 0;                                                                   // Untergrenze des Eigenzeit-Intervalls [a,b] in dem die Loesung berechnet werden soll
    double b = 800;                                                                 // Obergrenze des Intervalls [a,b]
```

In der main()-Funktion des Programmes werden am Anfang die nötigen Variablen definiert, die man für die Euler bzw. Runge-Kutta Ordnung vier Methode benötigt. Die nötigen sechs Anfangswerte $\alpha_i(\tau = 0)$, $i = 1, 2, ..6$ werden dem Beispiel des Jupyter Notebooks GeodSchwarzschildII.ipynb entnommen. Die Simulation der Bewegung des Probekörpers erfolgt in einem Eigenzeit-Intervall $[a, b] = [0, 800]$, wobei wir $N = 10000$ Gitterpunkte verwenden.

In der main()-Funktion des Programmes werden am Anfang die nötigen Variablen definiert, die man für die Euler bzw. Runge-Kutta Ordnung vier Methode benötigt. Die nötigen sechs Anfangswerte $\alpha_i(\tau = 0)$, $i = 1, 2, ..6$ werden dem Beispiel des Jupyter Notebooks GeodSchwarzschildII.ipynb entnommen. Die Simulation der Bewegung des Probekörpers erfolgt in einem Eigenzeit-Intervall $[a, b] = [0, 800]$, wobei wir $N = 10000$ Gitterpunkte verwenden.

Öffnen

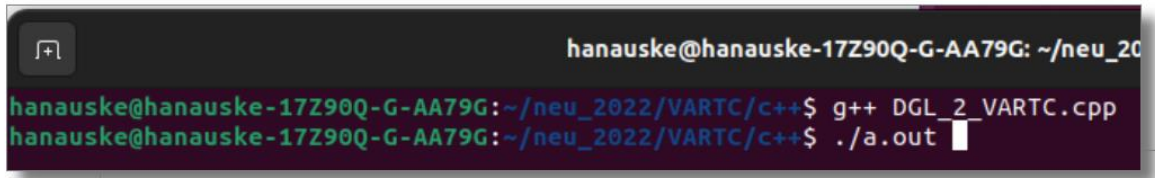
DGL_2_VARTC.cpp
~/neu_2022/VARTC/www/VARTC_2023

```
    double wert;
    wert = -2*y_5*y_6/y_2;                                  // Eigentliche Definition der Funktion
    return wert;                                            // Rueckgabewert der Funktion f_6
}                                                           // Ende der Funktion f_6

int main(){                                                 // Hauptfunktion
    double tau;                                             // Aktueller Eigenzeit-Wert
    double a = 0;                                           // Untergrenze des Eigenzeit-Intervalls [a,b] in dem die Loesung berechnet werden soll
    double b = 800;                                         // Obergrenze des Intervalls [a,b]
    int N = 10000;                                          // Anzahl der Punkte in die das tau-Intervall aufgeteilt wird
    double h = (b - a)/N;                                   // Abstand dtau zwischen den aequidistanten Punkten des tau-Intervalls (h=dtau)

    double alpha_1 = 0.0;                                   // 1.Anfangswert bei tau=a: y_1(a)=alpha_1 (entspricht Anfangswert der Koordinatenzeit t)
    double alpha_2 = 10.0;                                  // 2.Anfangswert bei tau=a: y_2(a)=alpha_2 (entspricht Anfangswert des Radius r)
    double alpha_3 = 0.0;                                   // 3.Anfangswert bei tau=a: y_3(a)=alpha_3 (entspricht Anfangswert des Winkels phi)
    double alpha_4 = 1.208356321620407;                     // 4.Anfangswert bei tau=a: y_4(a)=alpha_4 (entspricht Anfangswert dt/dtau)
    double alpha_5 = 0.0;                                   // 5.Anfangswert bei tau=a: y_5(a)=alpha_5 (entspricht Anfangswert dr/dtau)
    double alpha_6 = 0.041;                                 // 6.Anfangswert bei tau=a: y_6(a)=alpha_6 (entspricht Anfangswert dphi/dtau)

    double y_Euler_1 = alpha_1;                             // Deklaration und Initialisierung der numerischen Loesung der Euler Methode fuer y_1
    double y_Euler_2 = alpha_2;                             // Deklaration und Initialisierung der numerischen Loesung der Euler Methode fuer y_2
    double y_Euler_3 = alpha_3;                             // Deklaration und Initialisierung der numerischen Loesung der Euler Methode fuer y_3
    double y_Euler_4 = alpha_4;                             // Deklaration und Initialisierung der numerischen Loesung der Euler Methode fuer y_4
    double y_Euler_5 = alpha_5;                             // Deklaration und Initialisierung der numerischen Loesung der Euler Methode fuer y_5
    double y_Euler_6 = alpha_6;                             // Deklaration und Initialisierung der numerischen Loesung der Euler Methode fuer y_6

    double y_RungeK_4_1 = alpha_1;                          // Deklaration und Initialisierung der numerischen Loesung der Runge-Kutta Ordnung vier Methode
    double y_RungeK_4_2 = alpha_2;                          // Deklaration und Initialisierung der numerischen Loesung der Runge-Kutta Ordnung vier Methode
    double y_RungeK_4_3 = alpha_3;                          // Deklaration und Initialisierung der numerischen Loesung der Runge-Kutta Ordnung vier Methode
    double y_RungeK_4_4 = alpha_4;                          // Deklaration und Initialisierung der numerischen Loesung der Runge-Kutta Ordnung vier Methode
    double y_RungeK_4_5 = alpha_5;                          // Deklaration und Initialisierung der numerischen Loesung der Runge-Kutta Ordnung vier Methode
    double y_RungeK_4_6 = alpha_6;                          // Deklaration und Initialisierung der numerischen Loesung der Runge-Kutta Ordnung vier Methode

    double k1_1,k2_1,k3_1,k4_1;                             // Deklaration der vier Runge-Kutta Parameter fuer y_1
    double k1_2,k2_2,k3_2,k4_2;                             // Deklaration der vier Runge-Kutta Parameter fuer y_2
    double k1_3,k2_3,k3_3,k4_3;                             // Deklaration der vier Runge-Kutta Parameter fuer y_3
    double k1_4,k2_4,k3_4,k4_4;                             // Deklaration der vier Runge-Kutta Parameter fuer y_4
    double k1_5,k2_5,k3_5,k4_5;                             // Deklaration der vier Runge-Kutta Parameter fuer y_5
    double k1_6,k2_6,k3_6,k4_6;                             // Deklaration der vier Runge-Kutta Parameter fuer y_6

    double tmp_1,tmp_2,tmp_3;                               // Variablen zum Zwischenspeichern von Ergebnissen der Euler Methode

    printf("# 0: Index i \n# 1: tau-Wert \n# 2: Euler Methode y1 \n# 3: Euler Methode y2 \n# 4: Euler Methode y3 \n");    // Beschreibung der ausgegebenen Groessen
    printf("# 5: Runge-Kutta Ordnung vier y1 \n# 6: Runge-Kutta Ordnung vier y2 \n# 7: Runge-Kutta Ordnung vier y3 \n"); // Beschreibung der ausgegebenen Groessen
    printf("# 8: Runge-Kutta Ordnung vier y4 \n# 9: Runge-Kutta Ordnung vier y5 \n# 10: Runge-Kutta Ordnung vier y6 \n"); // Beschreibung der ausgegebenen Groessen

    for(int i=0; i <= N; ++i){                                                                                          // for-Schleife ueber die einzelnen Punkte des tau-Intervalls
        tau = a + i*h;                                                                                                  // Eigenzeit wird um h erhoeht
        printf("%3d %19.15f %19.15f %19.15f %19.15f %19.15f %19.15f",i, tau, y_Euler_1, y_Euler_2, y_Euler_3, y_RungeK_4_1, y_RungeK_4_2); // Ausgaben der Loesungen
        printf("%19.15f %19.15f %19.15f %19.15f \n",y_RungeK_4_3, y_RungeK_4_4,y_RungeK_4_5, y_RungeK_4_6);          // Ausgaben der Loesungen
```

Die Beschreibung der während der Simulation im Terminal ausgegebenen Größen wird au-erhalb der Berechnung-Schleife (for-Schleife) gemacht. Die for-Schleife verläuft über die einzelnen τ-Gitterpunkte und verwendet die Euler bzw. Runge-Kutta Ordnung vier Methode zum Lösen des DGL-Systems.

Die Beschreibung der während der Simulation im Terminal ausgegebenen Größen wird au-erhalb der Berechnung-Schleife (for-Schleife) gemacht. Die for-Schleife verläuft über die einzelnen τ-Gitterpunkte und verwendet die Euler bzw. Runge-Kutta Ordnung vier Methode zum Lösen des DGL-Systems.

```
    printf("# 5: Runge-Kutta Ordnung vier y1 \n# 6: Runge-Kutta Ordnung vier y2 \n# 7: Runge-Kutta Ordnung vier y3 \n");  // Beschreibung der ausgegebenen Groessen
    printf("# 8: Runge-Kutta Ordnung vier y4 \n# 9: Runge-Kutta Ordnung vier y5 \n# 10: Runge-Kutta Ordnung vier y6 \n"); // Beschreibung der ausgegebenen Groessen

    for(int i=0; i <= N; ++i){                                                                                          // for-Schleife ueber die einzelnen Punkte des tau-Intervalls
        tau = a + i*h;                                                                                                  // Eigenzeit wird um h erhoeht
        printf("%3d %19.15f %19.15f %19.15f %19.15f %19.15f %19.15f",i, tau, y_Euler_1, y_Euler_2, y_Euler_3, y_RungeK_4_1, y_RungeK_4_2); // Ausgaben der Loesungen
        printf("%19.15f %19.15f %19.15f %19.15f \n",y_RungeK_4_3, y_RungeK_4_4,y_RungeK_4_5, y_RungeK_4_6);              // Ausgaben der Loesungen

        tmp_1 = y_Euler_1 + h*f_1(y_Euler_1,y_Euler_2,y_Euler_3,y_Euler_4,y_Euler_5,y_Euler_6);     // Euler Methode fuer y_1
        tmp_2 = y_Euler_2 + h*f_2(y_Euler_1,y_Euler_2,y_Euler_3,y_Euler_4,y_Euler_5,y_Euler_6);     // Euler Methode fuer y_2
        tmp_3 = y_Euler_3 + h*f_3(y_Euler_1,y_Euler_2,y_Euler_3,y_Euler_4,y_Euler_5,y_Euler_6);     // Euler Methode fuer y_3
        y_Euler_4 = y_Euler_4 + h*f_4(y_Euler_1,y_Euler_2,y_Euler_3,y_Euler_4,y_Euler_5,y_Euler_6); // Euler Methode fuer y_4
        y_Euler_5 = y_Euler_5 + h*f_5(y_Euler_1,y_Euler_2,y_Euler_3,y_Euler_4,y_Euler_5,y_Euler_6); // Euler Methode fuer y_5
        y_Euler_6 = y_Euler_6 + h*f_6(y_Euler_1,y_Euler_2,y_Euler_3,y_Euler_4,y_Euler_5,y_Euler_6); // Euler Methode fuer y_6
        y_Euler_1 = tmp_1;
        y_Euler_2 = tmp_2;
        y_Euler_3 = tmp_3;

        //Runge-Kutta Ordnung vier Methode ...
        k1_1 = h*f_1(y_RungeK_4_1,y_RungeK_4_2,y_RungeK_4_3,y_RungeK_4_4,y_RungeK_4_5,y_RungeK_4_6);
        k1_2 = h*f_2(y_RungeK_4_1,y_RungeK_4_2,y_RungeK_4_3,y_RungeK_4_4,y_RungeK_4_5,y_RungeK_4_6);
        k1_3 = h*f_3(y_RungeK_4_1,y_RungeK_4_2,y_RungeK_4_3,y_RungeK_4_4,y_RungeK_4_5,y_RungeK_4_6);
        k1_4 = h*f_4(y_RungeK_4_1,y_RungeK_4_2,y_RungeK_4_3,y_RungeK_4_4,y_RungeK_4_5,y_RungeK_4_6);
        k1_5 = h*f_5(y_RungeK_4_1,y_RungeK_4_2,y_RungeK_4_3,y_RungeK_4_4,y_RungeK_4_5,y_RungeK_4_6);
        k1_6 = h*f_6(y_RungeK_4_1,y_RungeK_4_2,y_RungeK_4_3,y_RungeK_4_4,y_RungeK_4_5,y_RungeK_4_6);

        k2_1 = h*f_1(y_RungeK_4_1+k1_1/2,y_RungeK_4_2+k1_2/2,y_RungeK_4_3+k1_3/2,y_RungeK_4_4+k1_4/2,y_RungeK_4_5+k1_5/2,y_RungeK_4_6+k1_6/2);
        k2_2 = h*f_2(y_RungeK_4_1+k1_1/2,y_RungeK_4_2+k1_2/2,y_RungeK_4_3+k1_3/2,y_RungeK_4_4+k1_4/2,y_RungeK_4_5+k1_5/2,y_RungeK_4_6+k1_6/2);
        k2_3 = h*f_3(y_RungeK_4_1+k1_1/2,y_RungeK_4_2+k1_2/2,y_RungeK_4_3+k1_3/2,y_RungeK_4_4+k1_4/2,y_RungeK_4_5+k1_5/2,y_RungeK_4_6+k1_6/2);
        k2_4 = h*f_4(y_RungeK_4_1+k1_1/2,y_RungeK_4_2+k1_2/2,y_RungeK_4_3+k1_3/2,y_RungeK_4_4+k1_4/2,y_RungeK_4_5+k1_5/2,y_RungeK_4_6+k1_6/2);
        k2_5 = h*f_5(y_RungeK_4_1+k1_1/2,y_RungeK_4_2+k1_2/2,y_RungeK_4_3+k1_3/2,y_RungeK_4_4+k1_4/2,y_RungeK_4_5+k1_5/2,y_RungeK_4_6+k1_6/2);
        k2_6 = h*f_6(y_RungeK_4_1+k1_1/2,y_RungeK_4_2+k1_2/2,y_RungeK_4_3+k1_3/2,y_RungeK_4_4+k1_4/2,y_RungeK_4_5+k1_5/2,y_RungeK_4_6+k1_6/2);

        k3_1 = h*f_1(y_RungeK_4_1+k2_1/2,y_RungeK_4_2+k2_2/2,y_RungeK_4_3+k2_3/2,y_RungeK_4_4+k2_4/2,y_RungeK_4_5+k2_5/2,y_RungeK_4_6+k2_6/2);
        k3_2 = h*f_2(y_RungeK_4_1+k2_1/2,y_RungeK_4_2+k2_2/2,y_RungeK_4_3+k2_3/2,y_RungeK_4_4+k2_4/2,y_RungeK_4_5+k2_5/2,y_RungeK_4_6+k2_6/2);
        k3_3 = h*f_3(y_RungeK_4_1+k2_1/2,y_RungeK_4_2+k2_2/2,y_RungeK_4_3+k2_3/2,y_RungeK_4_4+k2_4/2,y_RungeK_4_5+k2_5/2,y_RungeK_4_6+k2_6/2);
        k3_4 = h*f_4(y_RungeK_4_1+k2_1/2,y_RungeK_4_2+k2_2/2,y_RungeK_4_3+k2_3/2,y_RungeK_4_4+k2_4/2,y_RungeK_4_5+k2_5/2,y_RungeK_4_6+k2_6/2);
        k3_5 = h*f_5(y_RungeK_4_1+k2_1/2,y_RungeK_4_2+k2_2/2,y_RungeK_4_3+k2_3/2,y_RungeK_4_4+k2_4/2,y_RungeK_4_5+k2_5/2,y_RungeK_4_6+k2_6/2);
        k3_6 = h*f_6(y_RungeK_4_1+k2_1/2,y_RungeK_4_2+k2_2/2,y_RungeK_4_3+k2_3/2,y_RungeK_4_4+k2_4/2,y_RungeK_4_5+k2_5/2,y_RungeK_4_6+k2_6/2);

        k4_1 = h*f_1(y_RungeK_4_1+k3_1,y_RungeK_4_2+k3_2,y_RungeK_4_3+k3_3,y_RungeK_4_4+k3_4,y_RungeK_4_5+k3_5,y_RungeK_4_6+k3_6);
        k4_2 = h*f_2(y_RungeK_4_1+k3_1,y_RungeK_4_2+k3_2,y_RungeK_4_3+k3_3,y_RungeK_4_4+k3_4,y_RungeK_4_5+k3_5,y_RungeK_4_6+k3_6);
        k4_3 = h*f_3(y_RungeK_4_1+k3_1,y_RungeK_4_2+k3_2,y_RungeK_4_3+k3_3,y_RungeK_4_4+k3_4,y_RungeK_4_5+k3_5,y_RungeK_4_6+k3_6);
        k4_4 = h*f_4(y_RungeK_4_1+k3_1,y_RungeK_4_2+k3_2,y_RungeK_4_3+k3_3,y_RungeK_4_4+k3_4,y_RungeK_4_5+k3_5,y_RungeK_4_6+k3_6);
        k4_5 = h*f_5(y_RungeK_4_1+k3_1,y_RungeK_4_2+k3_2,y_RungeK_4_3+k3_3,y_RungeK_4_4+k3_4,y_RungeK_4_5+k3_5,y_RungeK_4_6+k3_6);
        k4_6 = h*f_6(y_RungeK_4_1+k3_1,y_RungeK_4_2+k3_2,y_RungeK_4_3+k3_3,y_RungeK_4_4+k3_4,y_RungeK_4_5+k3_5,y_RungeK_4_6+k3_6);

        y_RungeK_4_1 = y_RungeK_4_1 + (k1_1 + 2*k2_1 + 2*k3_1 + k4_1)/6;
        y_RungeK_4_2 = y_RungeK_4_2 + (k1_2 + 2*k2_2 + 2*k3_2 + k4_2)/6;
        y_RungeK_4_3 = y_RungeK_4_3 + (k1_3 + 2*k2_3 + 2*k3_3 + k4_3)/6;
        y_RungeK_4_4 = y_RungeK_4_4 + (k1_4 + 2*k2_4 + 2*k3_4 + k4_4)/6;
        y_RungeK_4_5 = y_RungeK_4_5 + (k1_5 + 2*k2_5 + 2*k3_5 + k4_5)/6;
        y_RungeK_4_6 = y_RungeK_4_6 + (k1_6 + 2*k2_6 + 2*k3_6 + k4_6)/6;

    }                                                   // Ende for-Schleife ueber die einzelnen Punkte des tau-Intervalls
}                                                       // Ende der Hauptfunktion
```



An jedem Eigenzeit-Gitterpunkt werden die berechneten Größen im Terminal ausgegeben. Nach Kompilierung des Programms (siehe links nebenstehende Abbildung) erhält man die folgende Terminal-Ausgabe (siehe unteres Bild).

```
9977 798.159999999999968 908.073370856684392  11.344729392722476  18.935085074717669 906.485132018719355  10.943831231460884 18.608466320865247  1.182853069475079  0.050260777500912  0.034233012506669
9978 798.240000000000009 908.167242179071536  11.349072875435889  18.937622635777164 906.579756376174714  10.947855163685354 18.611203955522331  1.182755856785676  0.050337452504322  0.034207852152829
9979 798.320000000000050 908.261105809443052  11.353420321837079  18.940158251984506 906.674372954195064  10.951885220841859 18.613939576929546  1.182658583721394  0.050413900807322  0.034182681293164
9980 798.399999999999977 908.354961747931839  11.357771715597153  18.942691923810195 906.768981747979296  10.955921384795074 18.616673184254388  1.182561251000450  0.050490122428327  0.034157500126557
9981 798.480000000000018 908.448809994709450  11.362127040401063  18.945223651734672 906.863582752783486  10.959963637411230 18.619404776680234  1.182463859339188  0.050566117387832  0.034132308851276
9982 798.560000000000059 908.542650549985638  11.366486279947702  18.947753436248284 906.958175963920894  10.964011960558283 18.622134353406299  1.182366409452069  0.050641885708402  0.034107107664970
9983 798.639999999999986 908.636483414008239  11.370849417949987  18.950281277851222 907.052761376762078  10.968066336106077 18.624861913647589  1.182268902051671  0.050717427414665  0.034081896764673
9984 798.720000000000027 908.730308587063291  11.375216438134940  18.952807177053486 907.147338986734439  10.972126745926506 18.627587456634846  1.182171337848678  0.050792742533303  0.034056676346797
9985 798.800000000000068 908.824126069474460  11.379587324243783  18.955331134374838 907.241908789322110  10.976193171893682 18.630310981614496  1.182073717551880  0.050867831093041  0.034031446607133
9986 798.879999999999995 908.917935861603155  11.383962060032015  18.957853150344746 907.336470780065838  10.980265595884097 18.633032487848610  1.181976041868162  0.050942693124644  0.034006207740850
9987 798.960000000000036 909.011737963848418  11.388340629269502  18.960373225502352 907.431024954562872  10.984343999776783 18.635751974614845  1.181878311502501  0.051017328660902  0.033980959942494
9988 799.039999999999964 909.105532376646806  11.392723015740559  18.962891360396412 907.525571308466851  10.988428365453480 18.638469441206393  1.181780527157964  0.051091737736627  0.033955703405986
9989 799.120000000000005 909.199319100471826  11.397109203244034  18.965407555585269 907.620109837487462  10.992518674798790 18.641184886931939  1.181682689535698  0.051165920388640  0.033930438324619
9990 799.200000000000045 909.293098135834384  11.401499175593388  18.967921811636785 907.714640537390437  10.996614909700343 18.643898311115606  1.181584799334926  0.051239876655764  0.033905164891059
9991 799.279999999999973 909.386869483282226  11.405892916616784  18.970434129128307 907.809163403997559  11.000717052048955 18.646609713096904  1.181486857252946  0.051313606578819  0.033879883297345
9992 799.360000000000014 909.480633143999928  11.410290410157160  18.972944508646627 907.903678433185974  11.004825083738785 18.649319092230680  1.181388863985121  0.051387110200608  0.033854593734885
9993 799.440000000000055 909.574389116808675  11.414691640072316  18.975452950787925 907.998185620888648  11.008938986667497 18.652026447887074  1.181290820224876  0.051460387565909  0.033829296394455
9994 799.519999999999982 909.668137404166146  11.419096590234995  18.977959456157727 908.092684963093916  11.013058742736414 18.654731779451453  1.181192726663694  0.051533438721470  0.033803991466201
9995 799.600000000000023 909.761878006166512  11.423505244532958  18.980464025370857 908.187176455845247  11.017184333850681 18.657435086324377  1.181094583991112  0.051606263715999  0.033778679139634
9996 799.680000000000064 909.855610923539984  11.427917586869068  18.982966659051403 908.281660095241250  11.021315741919414 18.660136367921542  1.180996392894715  0.051678862600152  0.033753359603631
9997 799.759999999999991 909.949336157053040  11.432333601161366  18.985467357832658 908.376135877435445  11.025452948855863 18.662835623673725  1.180898154060132  0.051751235426529  0.033728033046434
9998 799.840000000000032 910.043053707507966  11.436753271343150  18.987966122357079 908.470603798636148  11.029595936577561 18.665532853026743  1.180799868171030  0.051823382249664  0.033702699655649
9999 799.920000000000073 910.136763575742862  11.441176581363056  18.990462953276246 908.565063855106246  11.033744687006481 18.668228055441389  1.180701535909115  0.051895303126013  0.033677359618245
10000 800.000000000000000 910.230465762631297  11.445603515185130  18.992957851250814 908.659516043163080  11.037899182069188 18.670921230393390  1.180603157954122  0.051966998113951  0.033652013120551
hanauske@hanauske-17Z90Q-G-AA79G:~/neu_2022/VARTC/www/VARTC_2023$
```

```
hanauske@hanauske-17Z90Q-G-AA79G: ~/neu_2022/VARTC/c++
hanauske@hanauske-17Z90Q-G-AA79G:~/neu_2022/VARTC/c++$ ./a.out > DGL_2_VARTC.dat
hanauske@hanauske-17Z90Q-G-AA79G:~/neu_2022/VARTC/c++$
```

Mittels des oberen Terminal-Befehls leiten wir die Ausgabe in die Datei 'DGL_2_VARTC.dat' um.

Das gesamte C++ Programm können Sie sich unter folgendem Link herunterladen:
DGL_2_VARTC.cpp

Resultate der Simulation

Beim Klicken auf das nebenstehende rechte Bild gelangen Sie zu dem Jupyter Notebook GeodSchwarzschildII.ipynb, in dem eine Visualisierung der Ergebnisse des oberen C++ Programms programmiert ist. Das Bild zeigt die Visualisierung der Daten von 'DGL_2_VARTC.dat' und vergleicht das Euler Verfahren mit der Runge-Kutta Ordnung vier Methode. Zusätzlich wird in der Abbildung auch die numerische Lösung, die in Python direkt generiert wurde (Methode "integrate.odeint()" im Python-Modul "scipy") mit den simulierten Daten der unterschiedlichen Verfahren verglichen.