Allgemeine Relativitätstheorie mit dem Computer

PC-POOL RAUM 01.120
JOHANN WOLFGANG GOETHE UNIVERSITÄT
05. JUNI, 2020

Aufgrund der Corona Krise findet die Vorlesung und die freiwilligen Übungstermine in diesem Semester nur Online statt.

MATTHIAS HANAUSKE

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

6. Vorlesung

Plan für die heutige Vorlesung

- Kerr-Metrik eines rotierenden schwarzen Loches, Ereignishorizonte und Flächen der stationären Grenze (bzw. der unendlichen Rotverschiebung), der Mitführungseffekt der Raumzeit ("Frame-Dragging"), der gravitomagnetische Effekt, geodätische Bewegung eines Probekörpers in der Kerr Metrik), Klassifizierung der möglichen Bahnbewegungen um ein rotierendes schwarzes Loch (Kerr Metrik) mittels eines effektiven Potentials, Kreisförmige Bewegungen in der äquatorialen Ebene
- Der "Innermost Stable Circular Orbit" für einen Probekörper der sich mit und entgegen der Rotationsrichtung des schwarzen Loches bewegt
- Einführung in die Parallele Programmierung

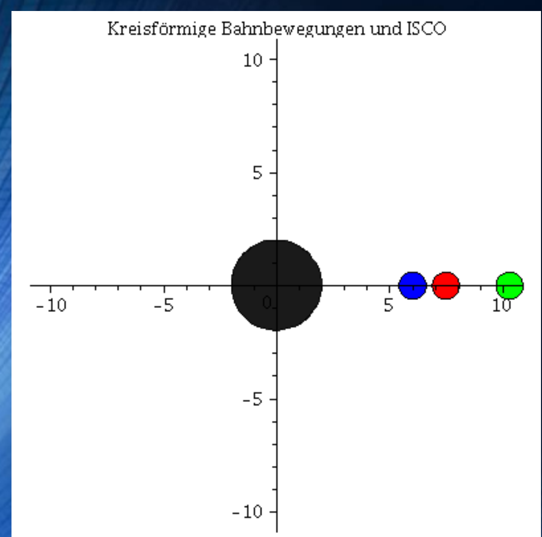
Geodätische Bewegung um ein nicht-rotierendes schwarzes Loch

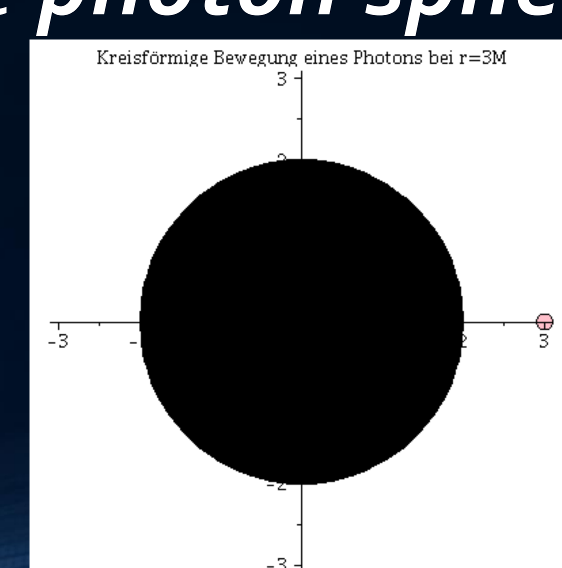
$$R_{\mu\nu} - \frac{1}{2} g_{\mu\nu} R = -8\pi T_{\mu\nu}$$

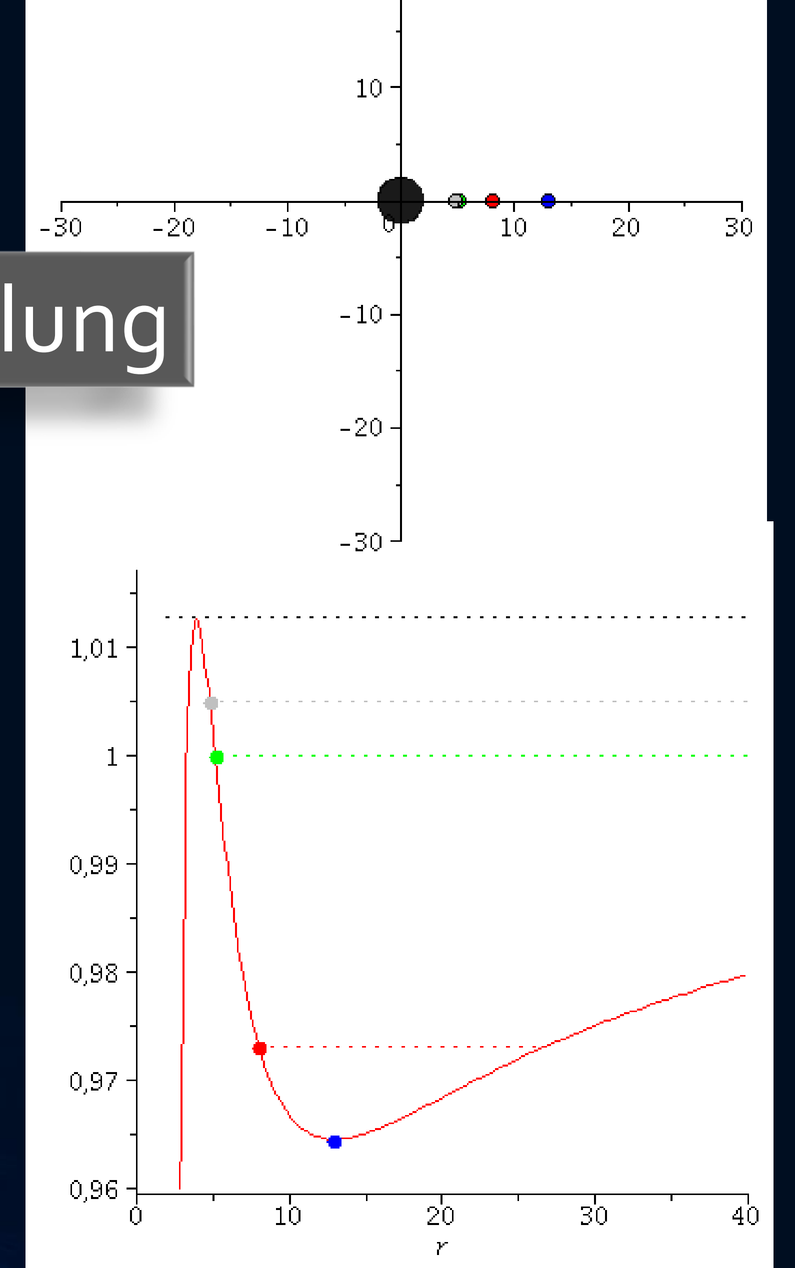
$$\frac{d^2 x^\mu}{d\tau^2} + \Gamma^\mu_{\nu\rho} \frac{dx^\nu}{d\tau} \frac{dx^\rho}{d\tau} = 0$$

Wiederholung

The *ISCO* and the *photon sphere*







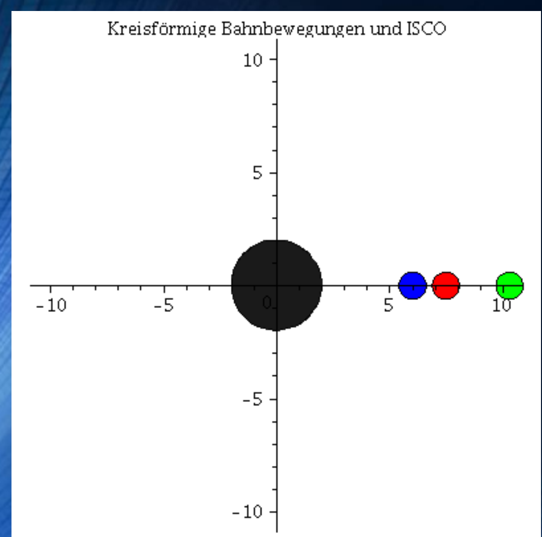
Geodätische Bewegung um ein nicht-rotierendes schwarzes Loch

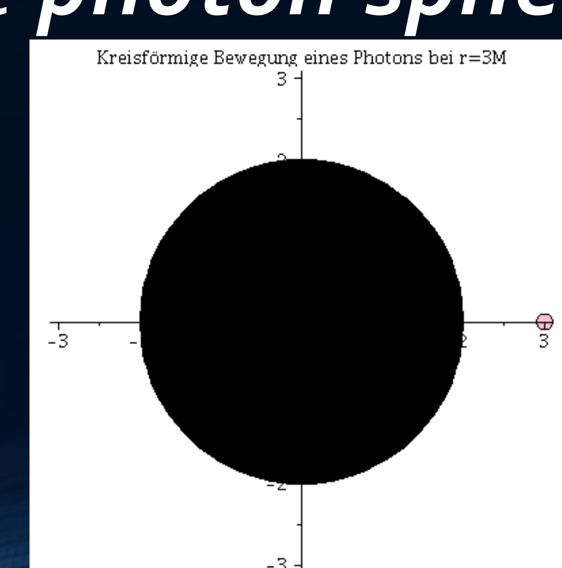
$$R_{\mu\nu} - \frac{1}{2} g_{\mu\nu} R = -8\pi T_{\mu\nu}$$

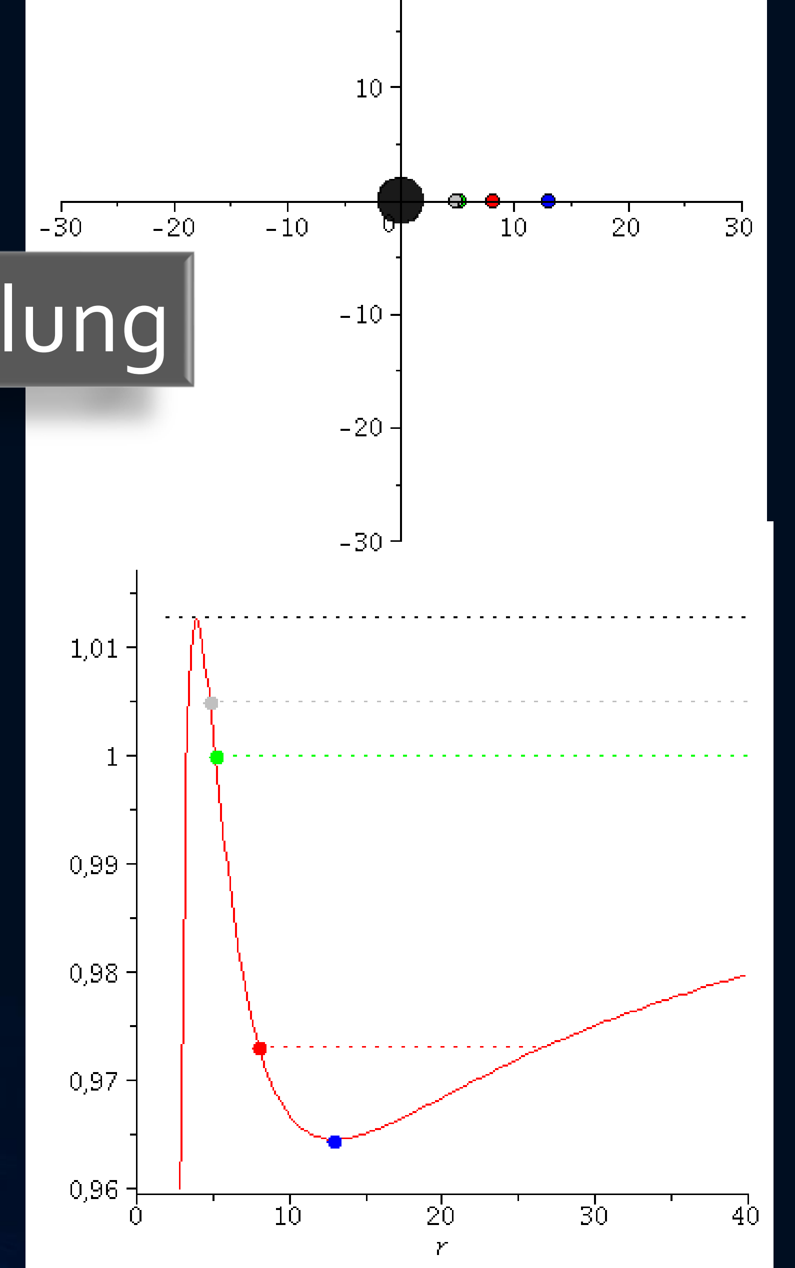
$$\frac{d^2 x^\mu}{d\tau^2} + \Gamma^\mu_{\nu\rho} \frac{dx^\nu}{d\tau} \frac{dx^\rho}{d\tau} = 0$$

Wiederholung

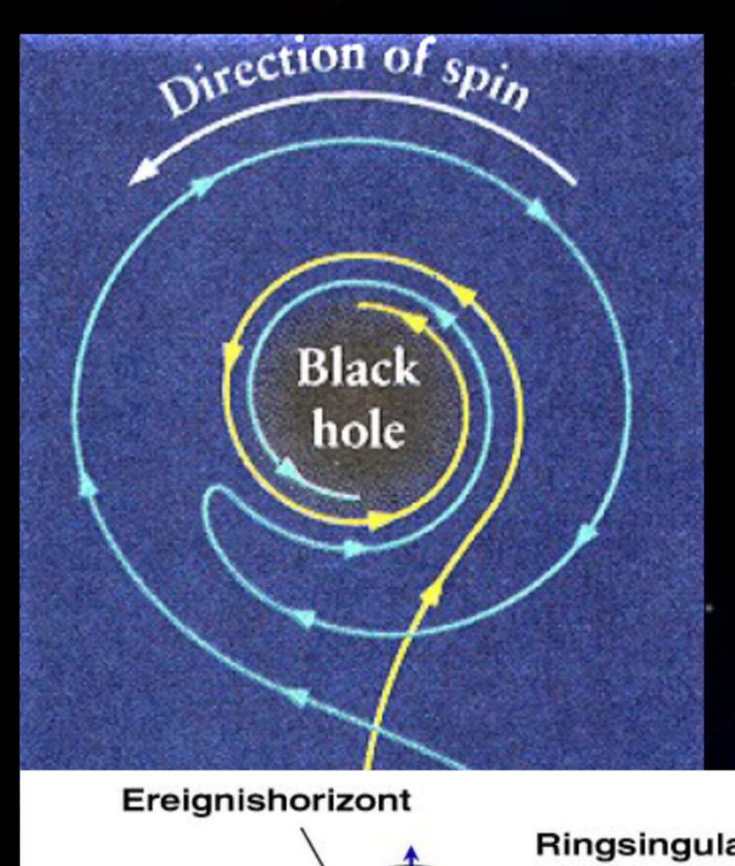
The *ISCO* and the ***photon sphere***

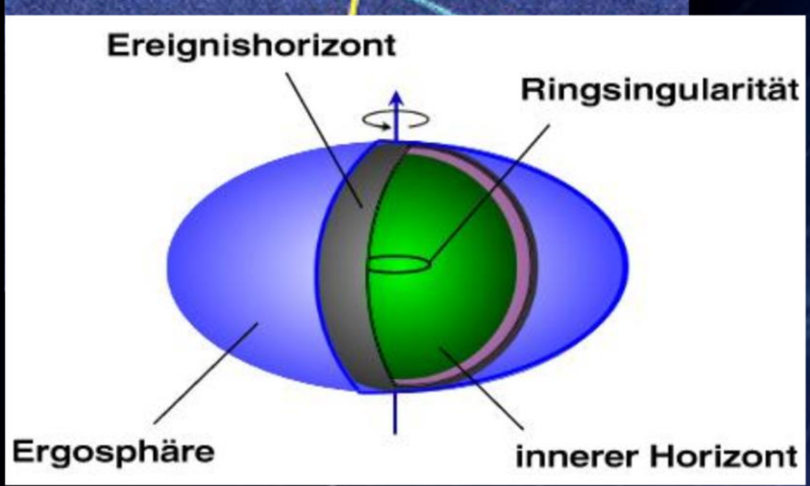






Rotierende schwarze Löcher und die Kerr Metrik

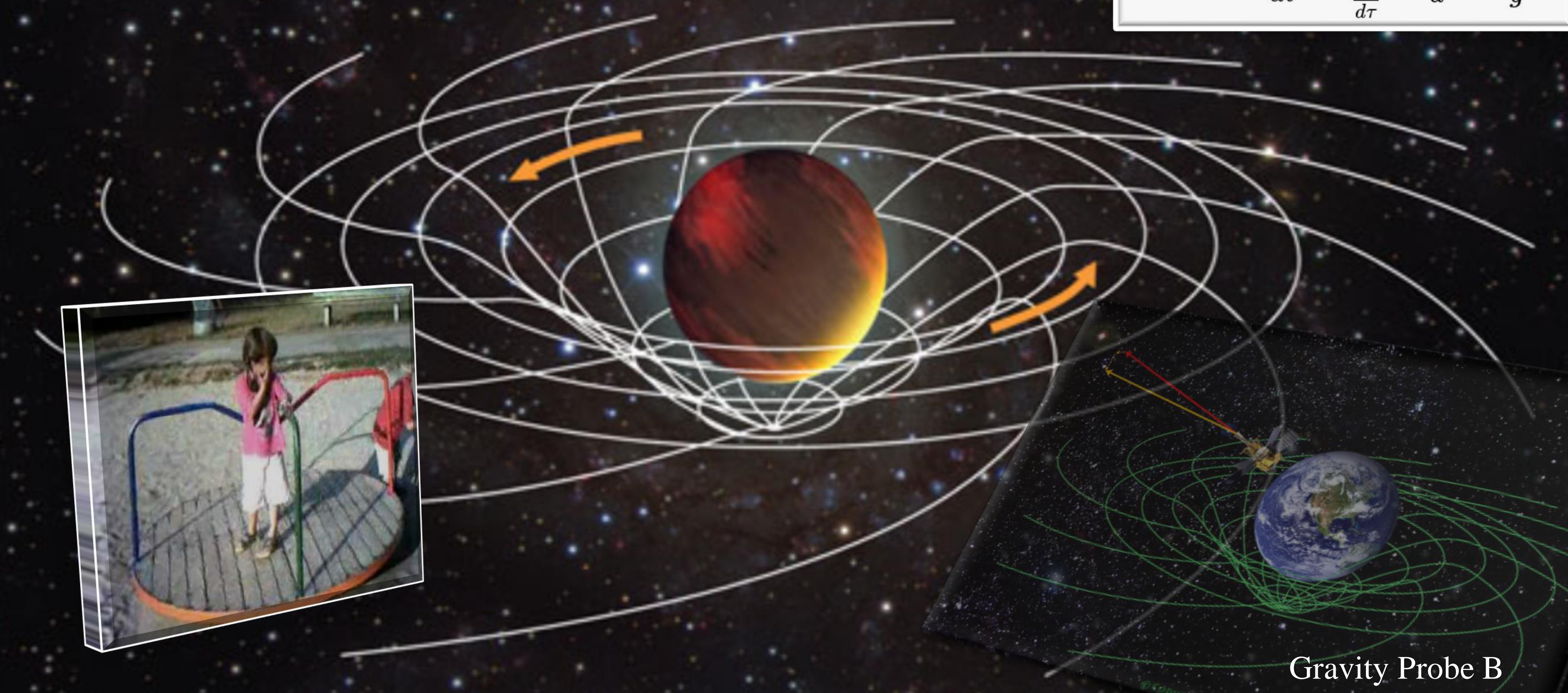




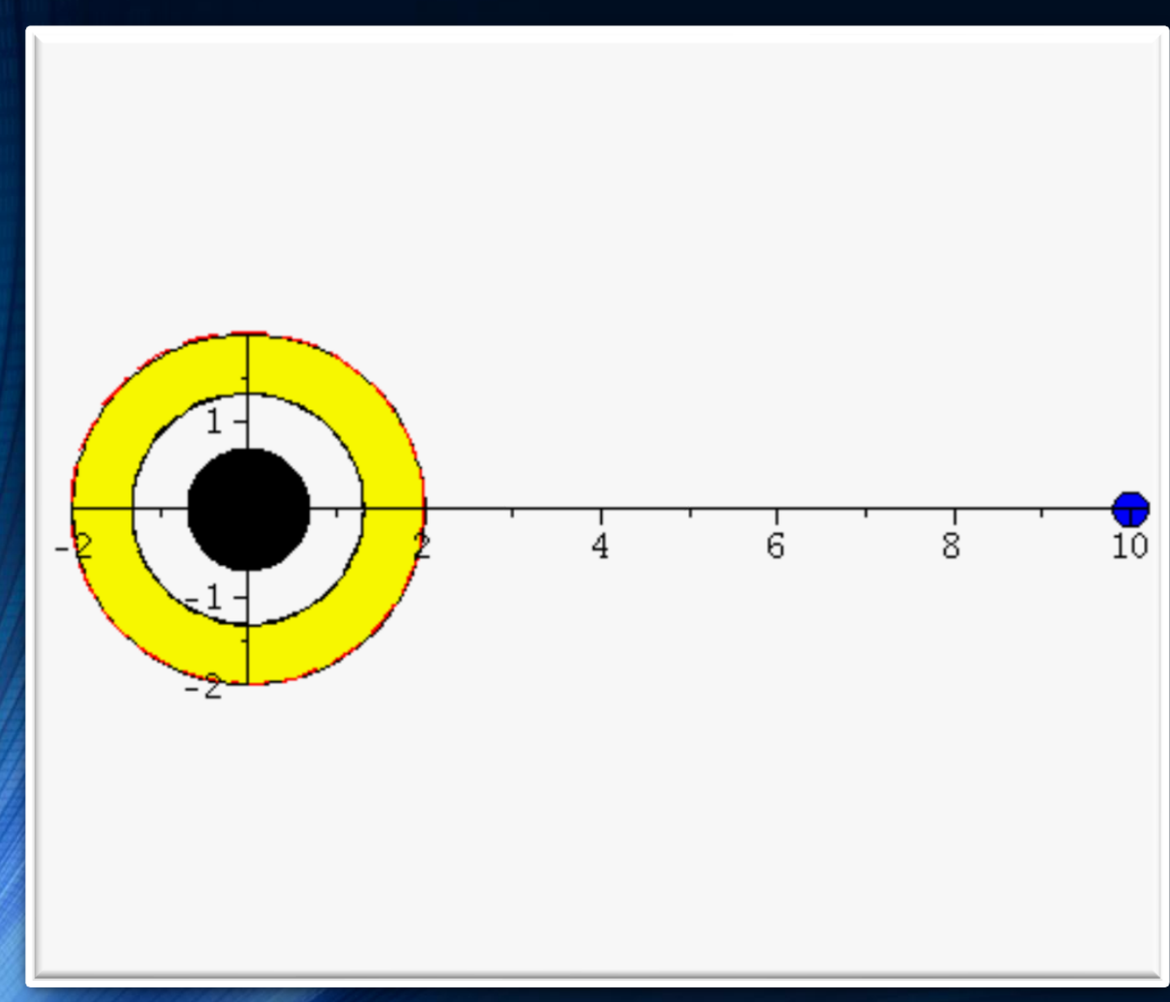
Der Mitführungseffekt der Raumzeit ("Frame-Dragging")

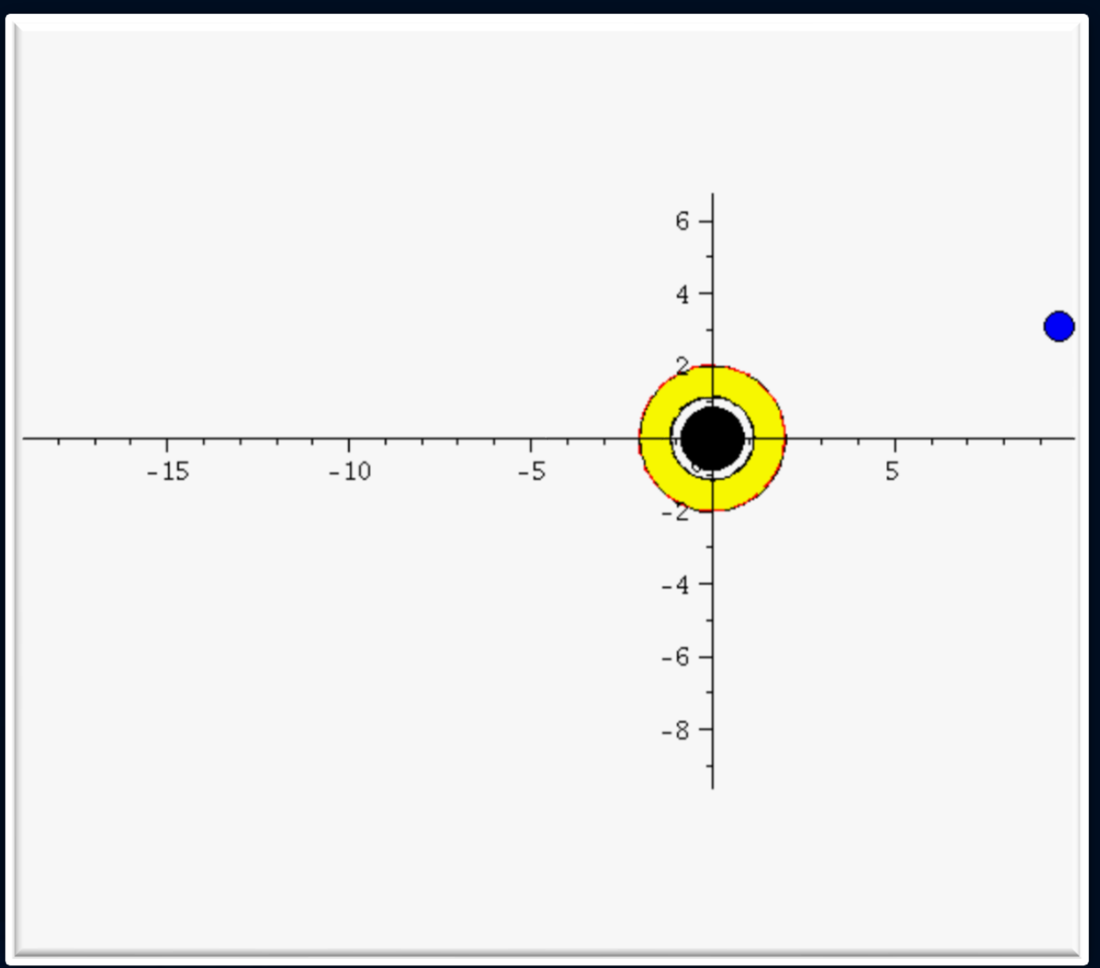
Experimente zur Bestätigung des Effektes: LARES, Gravity Probe B

$$\omega(r,\theta) = \frac{d\phi}{dt} = \frac{\frac{d\phi}{d\tau}}{\frac{dt}{d\tau}} = \frac{u^\phi}{u^t} = \frac{g^{t\phi}}{g^{tt}}$$



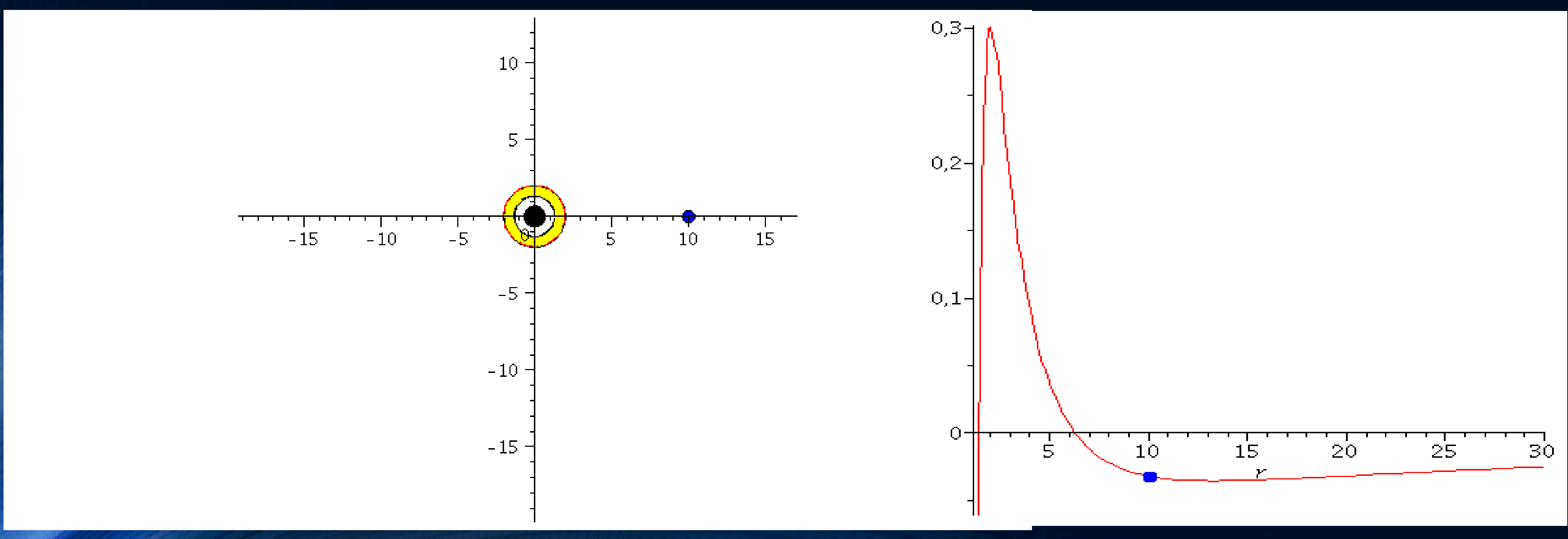
Der Mitführungseffekt der Raumzeit ("Frame-Dragging") und der Gravitomagnetische Effekt





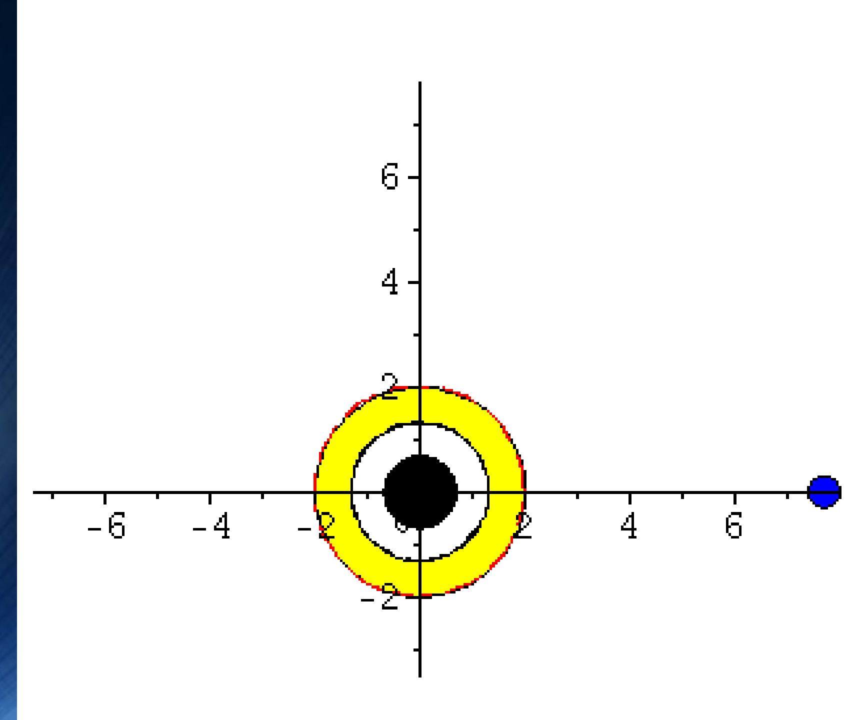
Kerr Metrik: Effektives Potential

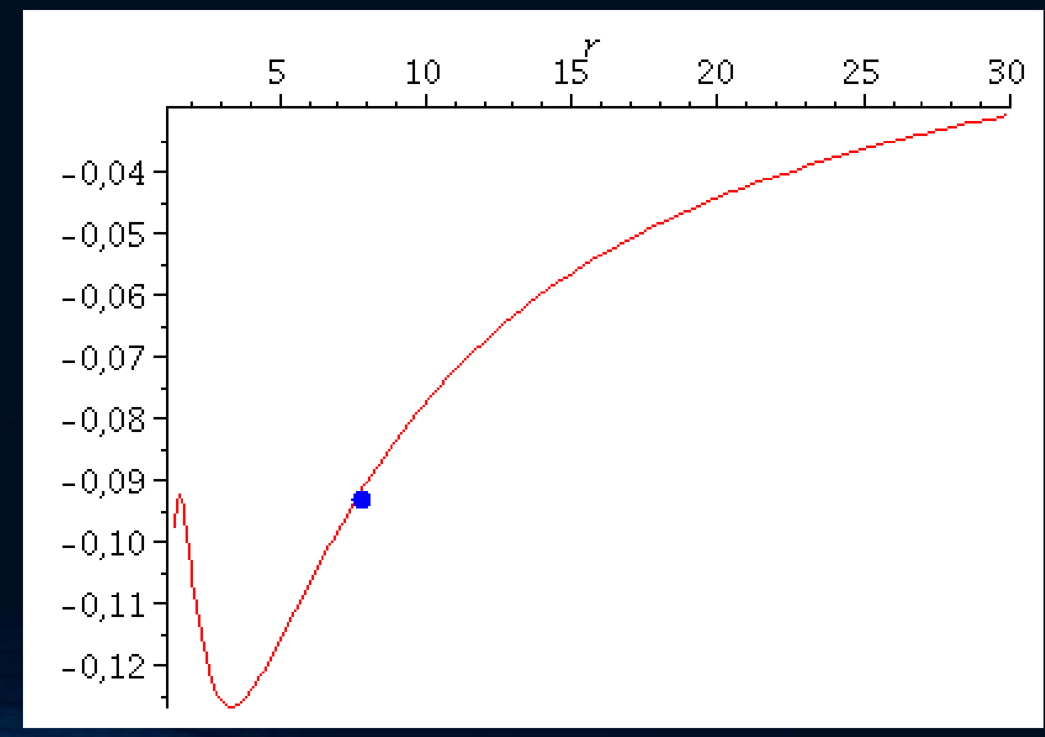
$$V_{eff}(r, M, l, a, E) = -\frac{M}{r} + \frac{l^2 - a^2\left(E^2 - 1\right)}{2\,r^2} - \frac{M(l - a\,E)^2}{r^3}$$



Kerr Metrik: Effektives Potential

$$V_{eff}(r, M, l, a, E) = -\frac{M}{r} + \frac{l^2 - a^2\left(E^2 - 1\right)}{2\,r^2} - \frac{M(l - a\,E)^2}{r^3}$$

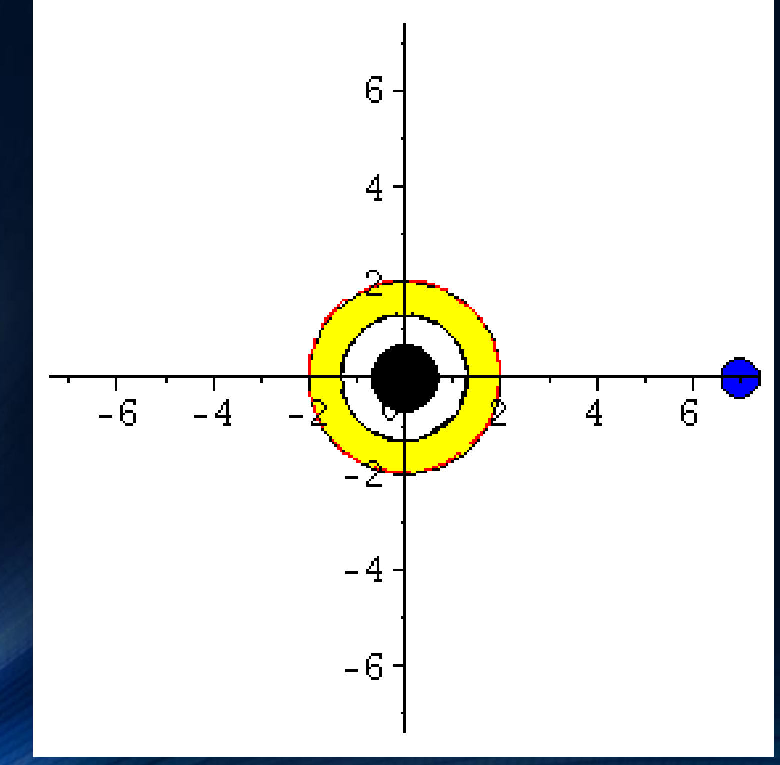


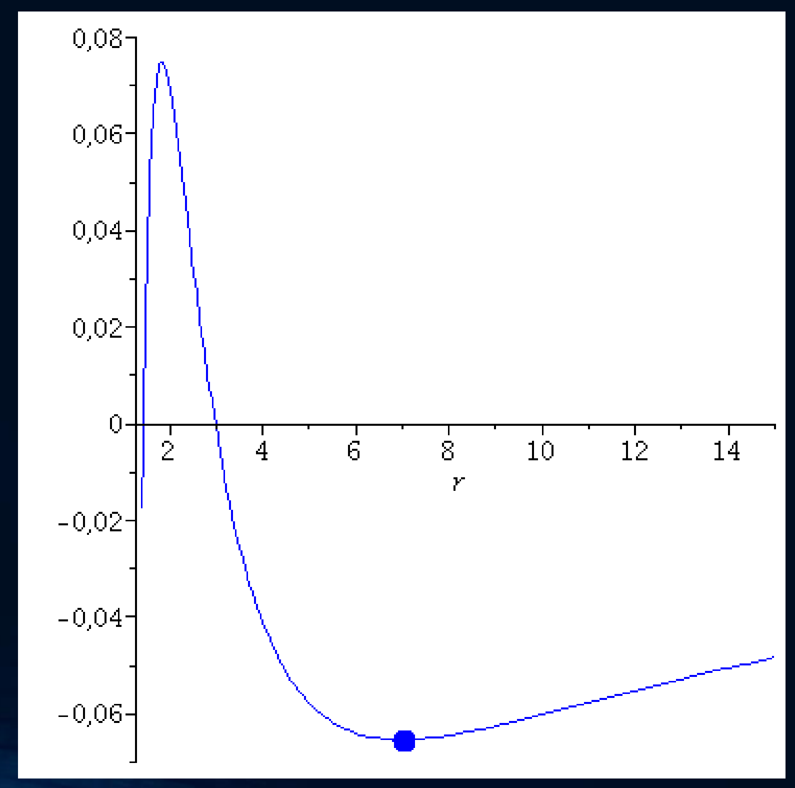


Kerr Metrik: Effektives Potential

Kreisförmige Bahnbewegungen

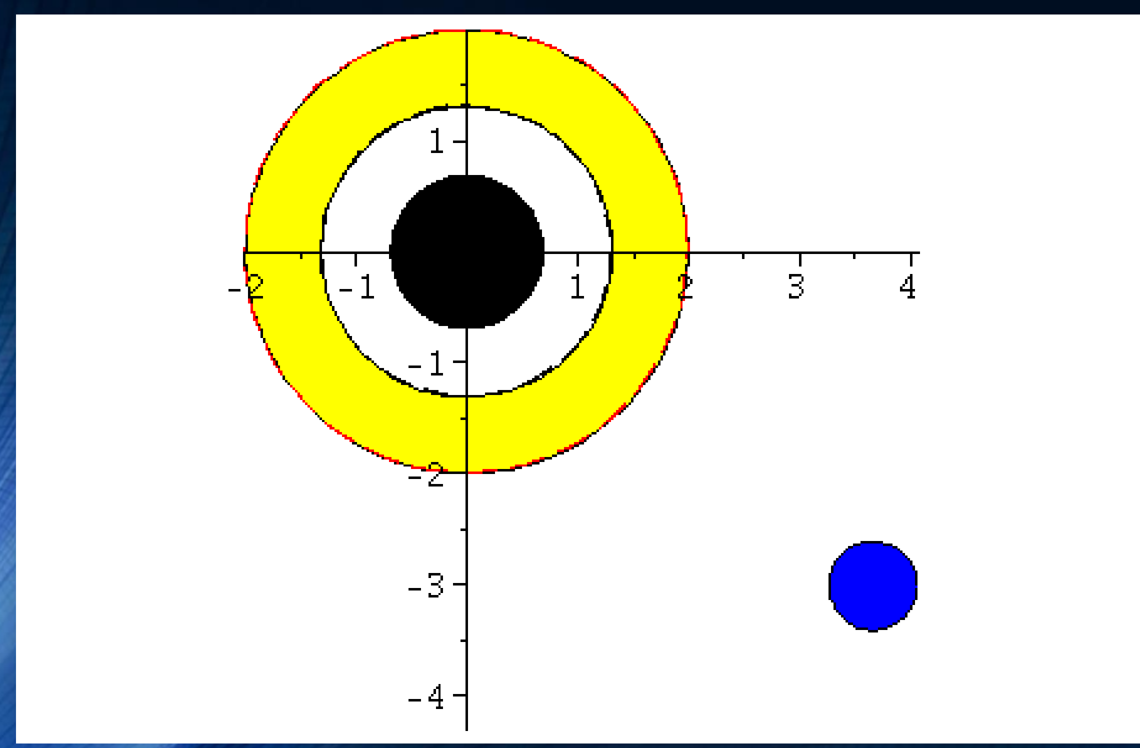
$$V_{eff}(r, M, l, a, E) = -\frac{M}{r} + \frac{l^2 - a^2\left(E^2 - 1\right)}{2\,r^2} - \frac{M(l - a\,E)^2}{r^3}$$

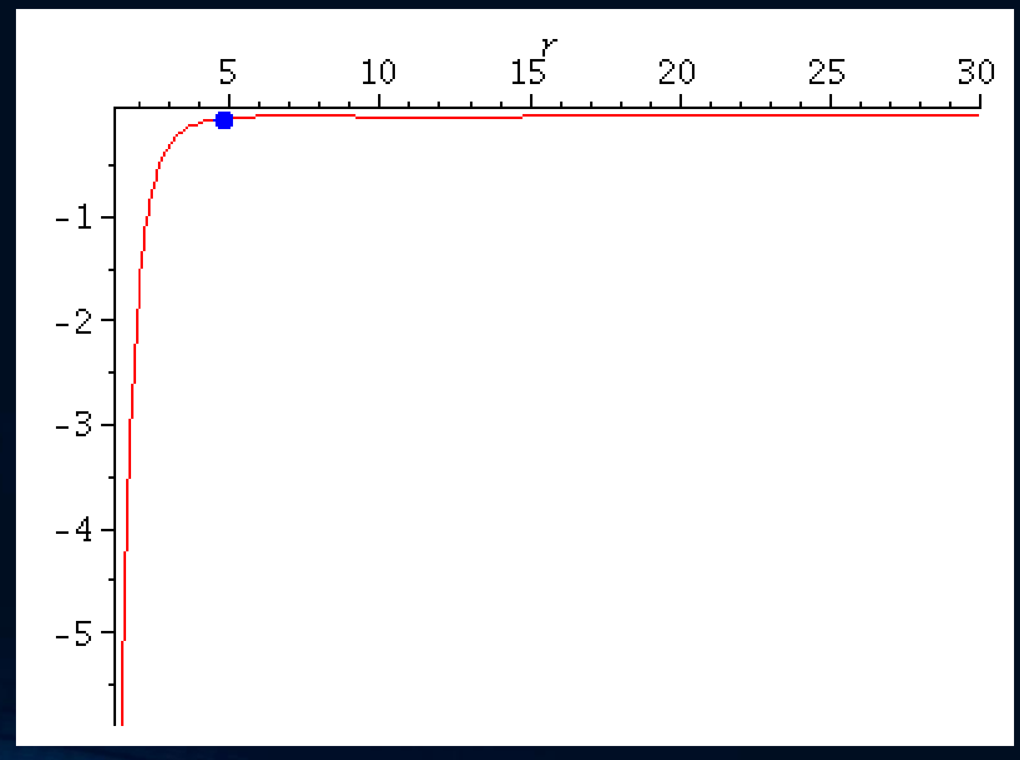


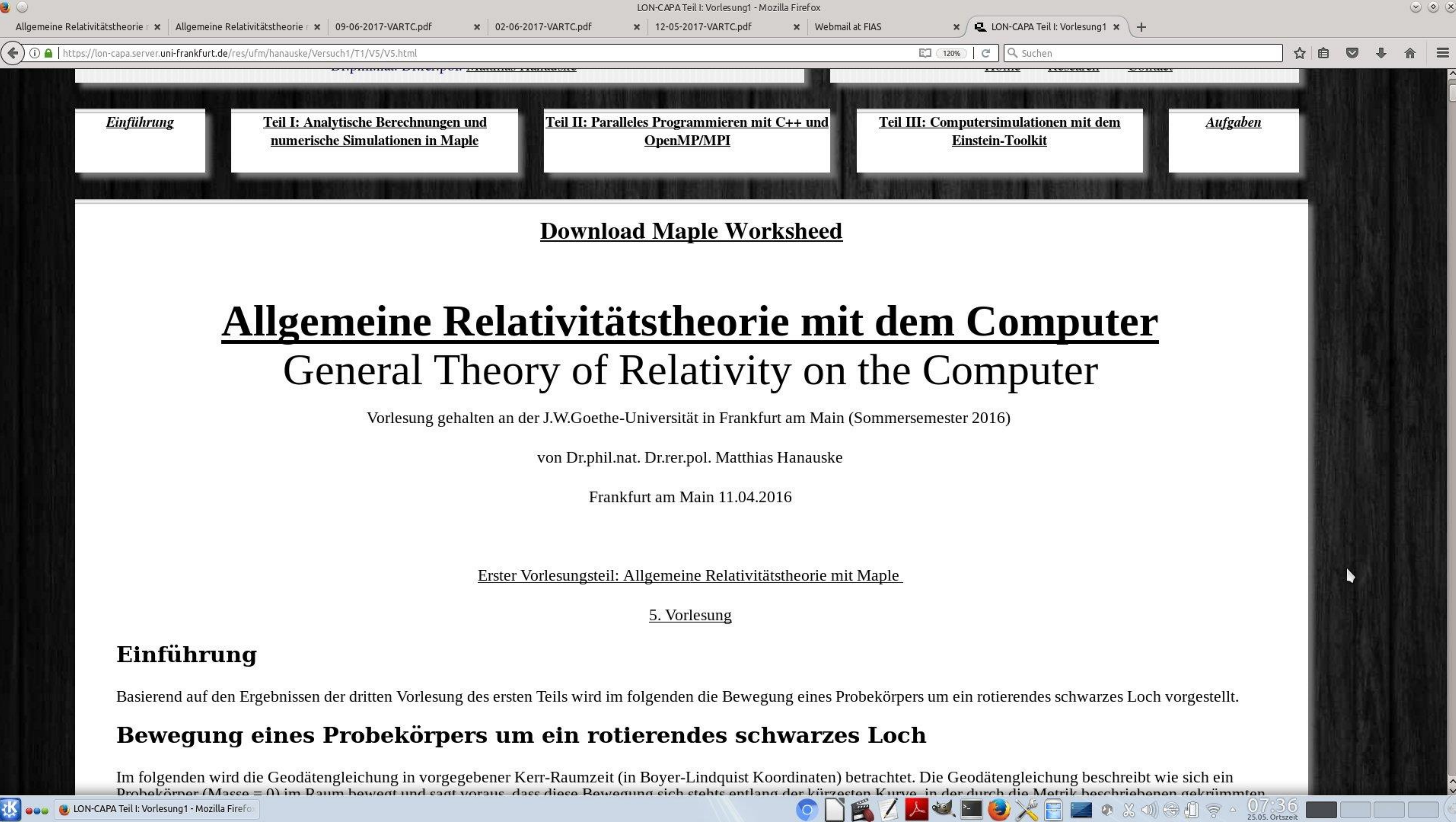


Kerr Metrik: Bewegungen innerhalb der Ergosphäre

$$V_{eff}(r, M, l, a, E) = -\frac{M}{r} + \frac{l^2 - a^2\left(E^2 - 1\right)}{2\,r^2} - \frac{M(l - a\,E)^2}{r^3}$$






LON-CAPA Teil I: Vorlesung1 - Mozilla Firefox
https://lon-capa.server.uni-frankfurt.de/res/ufm/hanauske/Versuch1/T1/V5/V5.html
Einführung
Teil I: Analytische Berechnungen und numerische Simulationen in Maple
Teil II: Paralleles Programmieren mit C++ und OpenMP/MPI
Teil III: Computersimulationen mit dem Einstein-Toolkit
Aufgaben
Download Maple Worksheed
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 2016)
von Dr.phil.nat. Dr.rer.pol. Matthias Hanauske
Frankfurt am Main 11.04.2016
Erster Vorlesungsteil: Allgemeine Relativitätstheorie mit Maple
5. Vorlesung
Einführung
Basierend auf den Ergebnissen der dritten Vorlesung des ersten Teils wird im folgenden die Bewegung eines Probekörpers um ein rotierendes schwarzes Loch vorgestellt.
Bewegung eines Probekörpers um ein rotierendes schwarzes Loch
Im folgenden wird die Geodätengleichung in vorgegebener Kerr-Raumzeit (in Boyer-Lindquist Koordinaten) betrachtet. Die Geodätengleichung beschreibt wie sich ein

LON-CAPA Teil I: Vorlesung1 - Mozilla Firefox
https://lon-capa.server.uni-frankfurt.de/res/ufm/hanauske/Versuch1/T1/V5/V5.html

Eigenschaften der Kerr-Metrik

```
>    restart:
  with( tensor ):
    with(plots):
 with(plottools):
```

Definition der kovarianten Raumzeit-Metrik eines rotierenden schwarzen Lochs der Masse M und Rotation a in Boyer-Lindquist Koordinaten (a ist ein spezifischer Drehimpuls a=J/M und wird als der sogenannte Kerr-Rotationsparameter bezeichnet). Die Kerr Metrik besitz folgendes Aussehen:

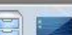
$$g_{\mu\nu} = \begin{pmatrix} g_{tt}(r,\theta) & 0 & 0 & g_{t\phi}(r,\theta) \\ 0 & g_{rr}(r,\theta) & 0 & 0 \\ 0 & 0 & g_{\theta\theta}(r,\theta) & 0 \\ g_{\phi t}(r,\theta) & 0 & 0 & g_{\phi\phi}(r,\theta) \end{pmatrix} \text{ , wobei:}$$

$$g_{tt}(r,\theta) = \left(\frac{1-2\,M\,r}{\rho^2}\right) \text{ , } g_{t\phi}(r,\theta) = \frac{2aMr\sin^2(\theta)}{\rho^2} \text{ , } g_{rr}(r,\theta) = -\frac{\rho^2}{\Delta} \text{ ,}$$

$$g_{\theta\theta}(r,\theta) = -\rho^2 \text{ , } g_{\phi\phi}(r,\theta) = -\left(\frac{r^2+a^2+2Mra^2\sin^2(\theta)}{\rho^2}\right)\sin^2(\theta) \text{ ,}$$

$$\rho^2 = r^2 + a^2\cos^2(\theta) \text{ , } \Delta = r^2 - 2Mr + a^2$$

VARTC2018 – Dolphin   LON-CAPA Teil I: Vorlesung1 - Mozilla Firefox



07:37
25.05. Ortszeit

LON-CAPA Teil I: Vorlesung1 - Mozilla Firefox
Allgemeine Relativitätstheorie | Allgemeine Relativitätstheorie | 09-06-2017-VARTC.pdf | 02-06-2017-VARTC.pdf | 12-05-2017-VARTC.pdf | Webmail at FIAS | LON-CAPA Teil I: Vorlesung1
https://lon-capa.server.uni-frankfurt.de/res/ufm/hanauske/Versuch1/T1/V5/V5.html

Struktur der Ereignishorizonte, Flächen der stationären Grenze und Flächen unendlicher Rotverschiebung

Die Flächen der stationären Grenze (stationary limit surfaces) und die der unendlichen Rotverschiebung sind durch $g_{tt} = 0$ bestimmt. Man erhält zwei Lösungen, die man gewöhnlich mit den Symbolen r_{S^+} und r_{S^-} bezeichnet.

```
> UnRot:=solve(get_compts(g)[1,1]=0,r);
```

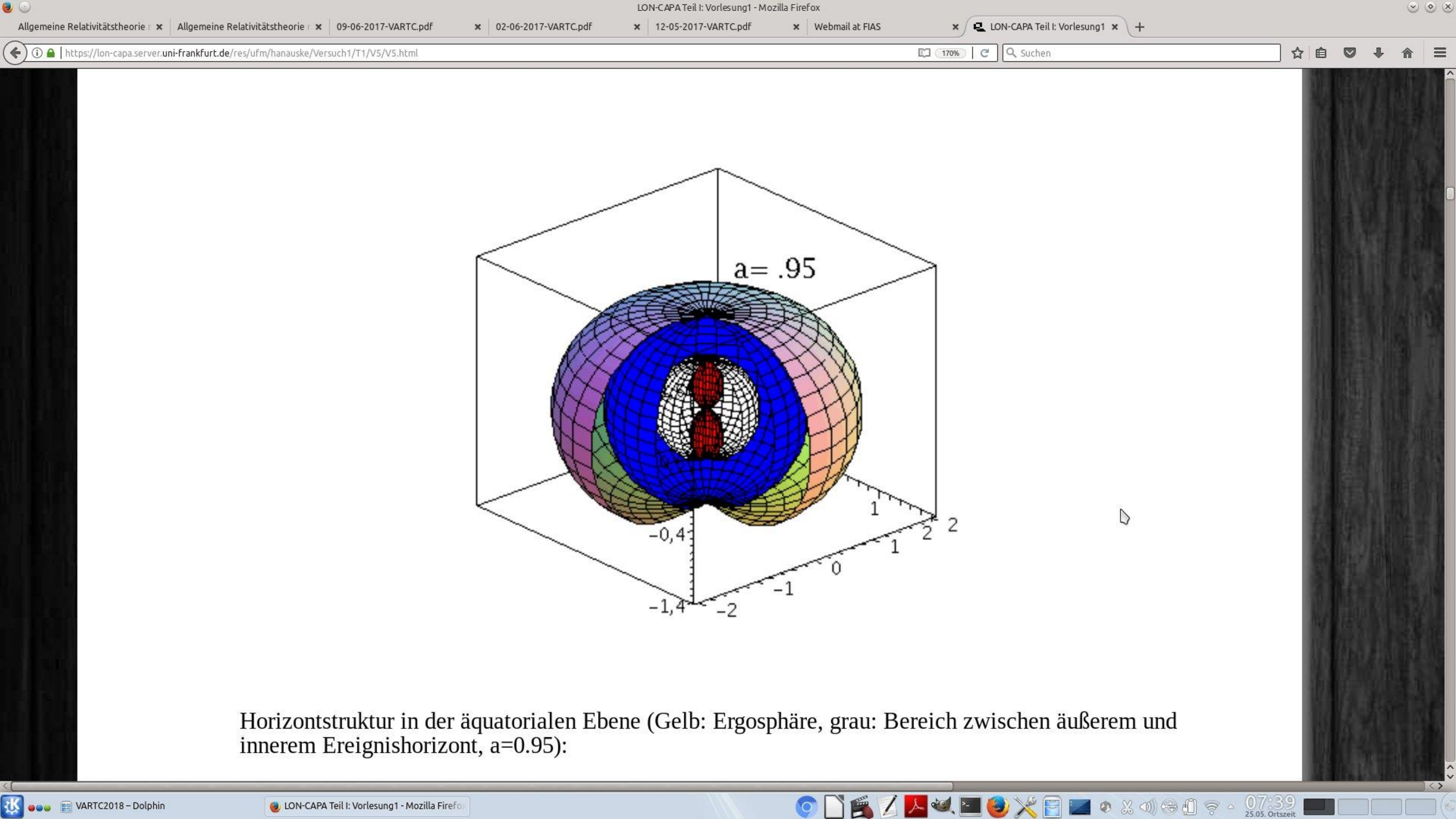
$$UnRot := M + \sqrt{M^2 - a^2 \cos(\theta)^2}, M - \sqrt{M^2 - a^2 \cos(\theta)^2} \quad (2.1.1.1)$$

Die Ereignishorizonte sind durch $g^{rr} = 0$ (bzw. $g_{rr} \to \infty$) bestimmt. Man erhält wieder zwei Lösungen, die man gewöhnlich mit den Symbolen r_+ und r_- bezeichnet.

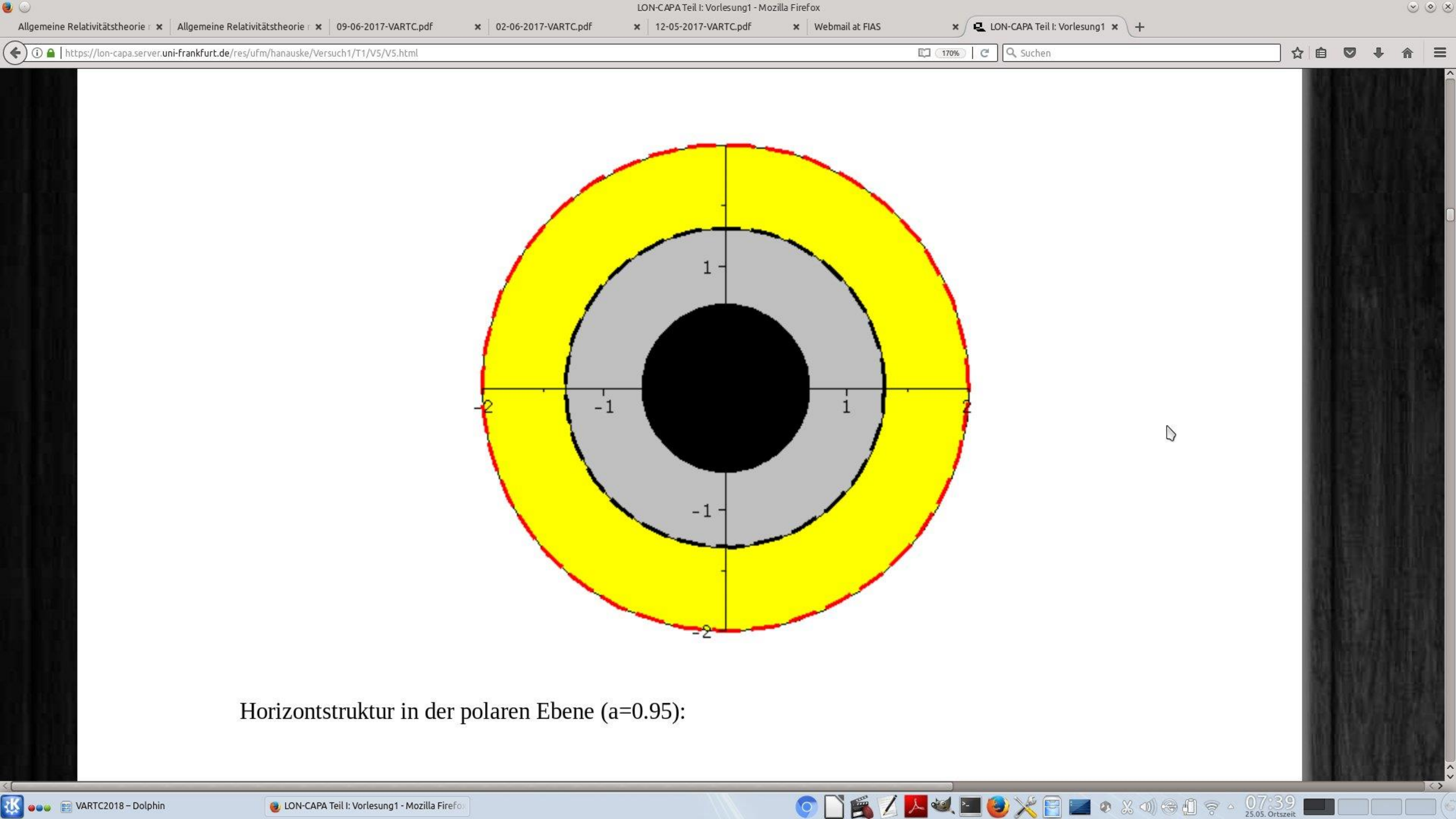
```
> Horizon:=solve(get_compts(ginv)[2,2]=0,r);
```

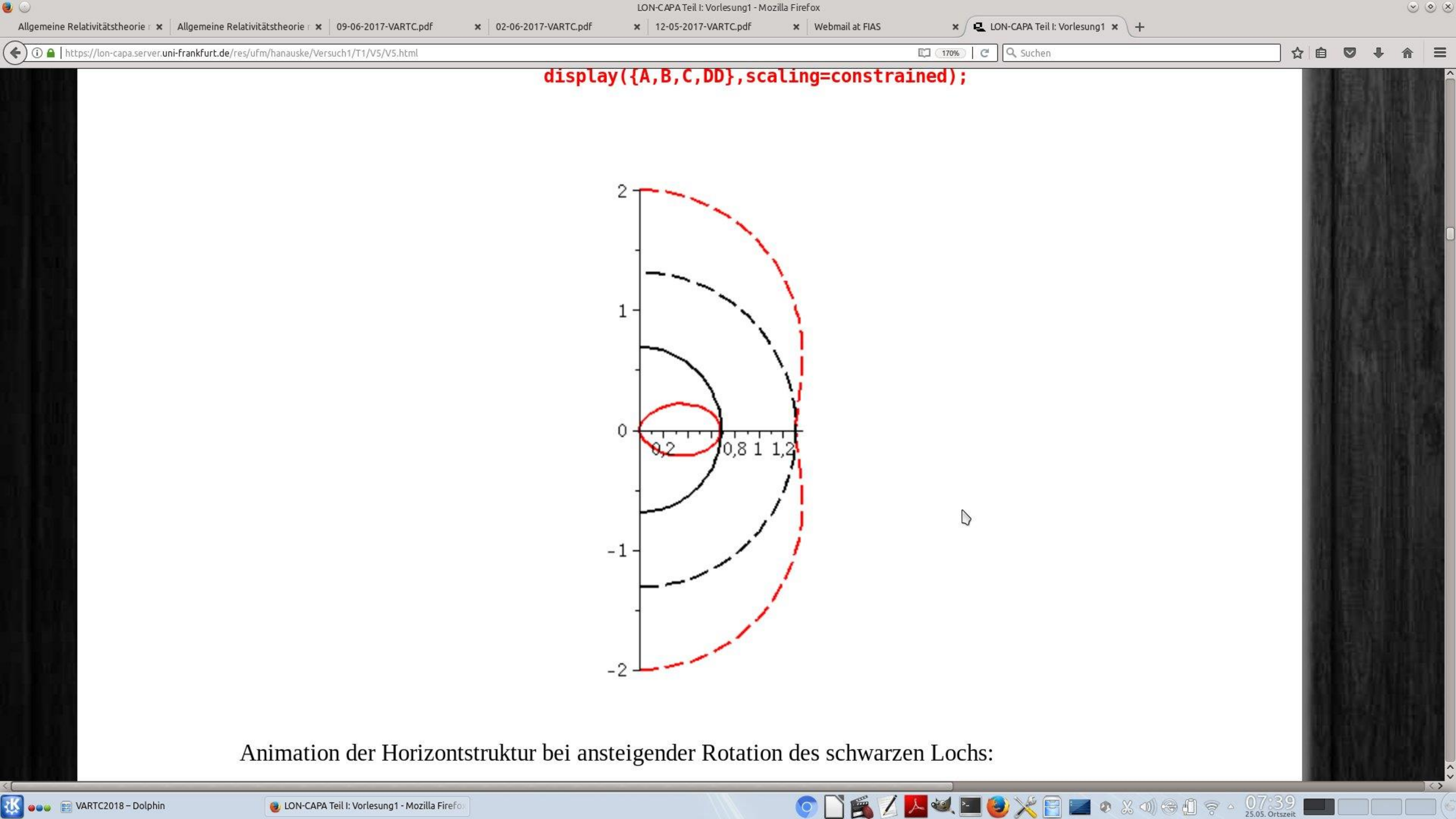
$$Horizon := M + \sqrt{M^2 - a^2}, M - \sqrt{M^2 - a^2} \quad (2.1.1.2)$$

VARTC2018 – Dolphin | LON-CAPA Teil I: Vorlesung1 - Mozilla Firefox | 07:38 25.05. Ortszeit

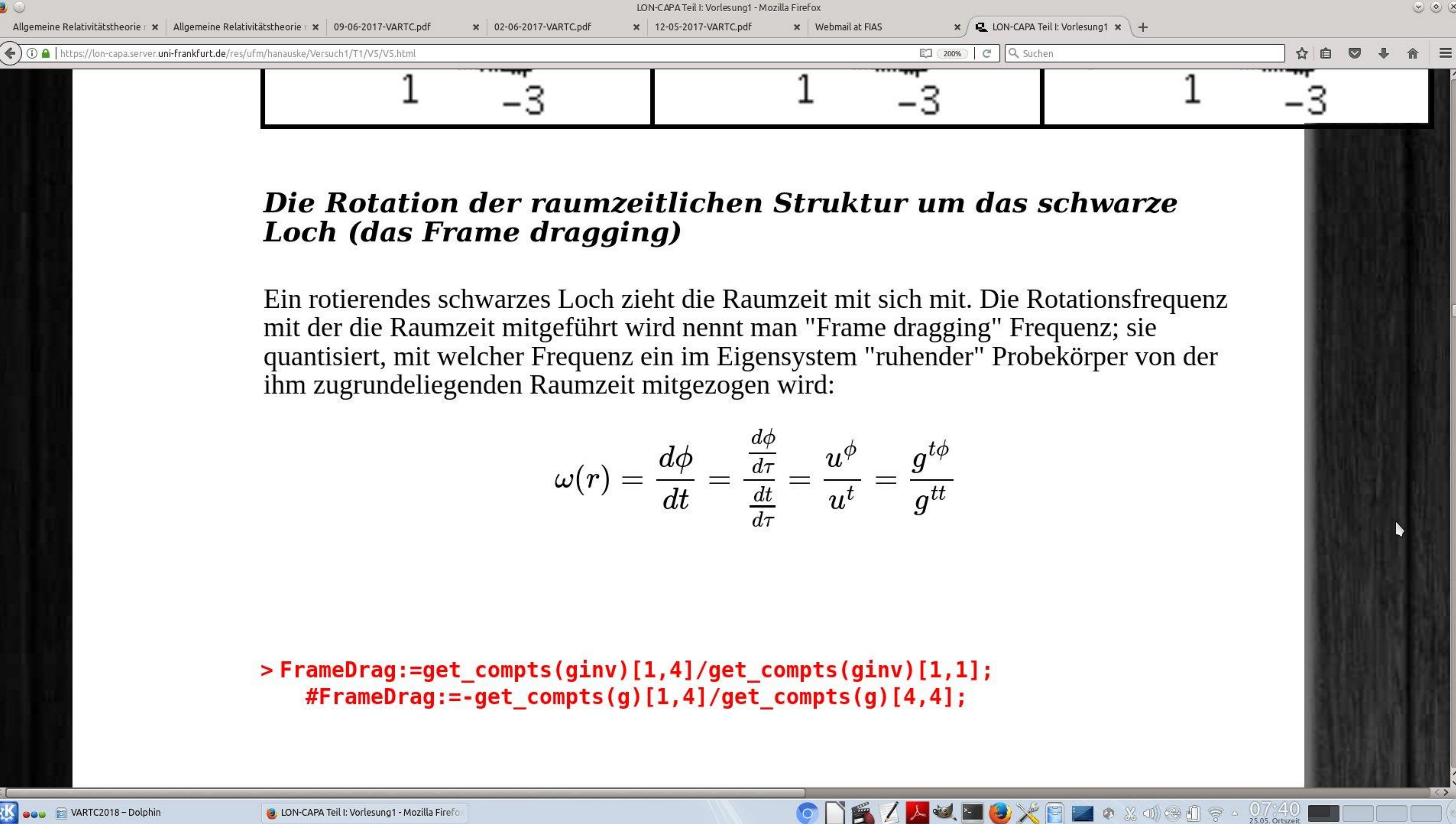


Horizontstruktur in der äquatorialen Ebene (Gelb: Ergosphäre, grau: Bereich zwischen äußerem und innerem Ereignishorizont, a=0.95):


LON-CAPA Teil I: Vorlesung1 - Mozilla Firefox
https://lon-capa.server.uni-frankfurt.de/res/ufm/hanauske/Versuch1/T1/V5/V5.html
1
-1
-2
-1
1
2
-1
-2
Horizontstruktur in der polaren Ebene (a=0.95):



display({A,B,C,DD},scaling=constrained);
2
1
0
-1
-2
0,2
0,8
1
1,2
Animation der Horizontstruktur bei ansteigender Rotation des schwarzen Lochs:

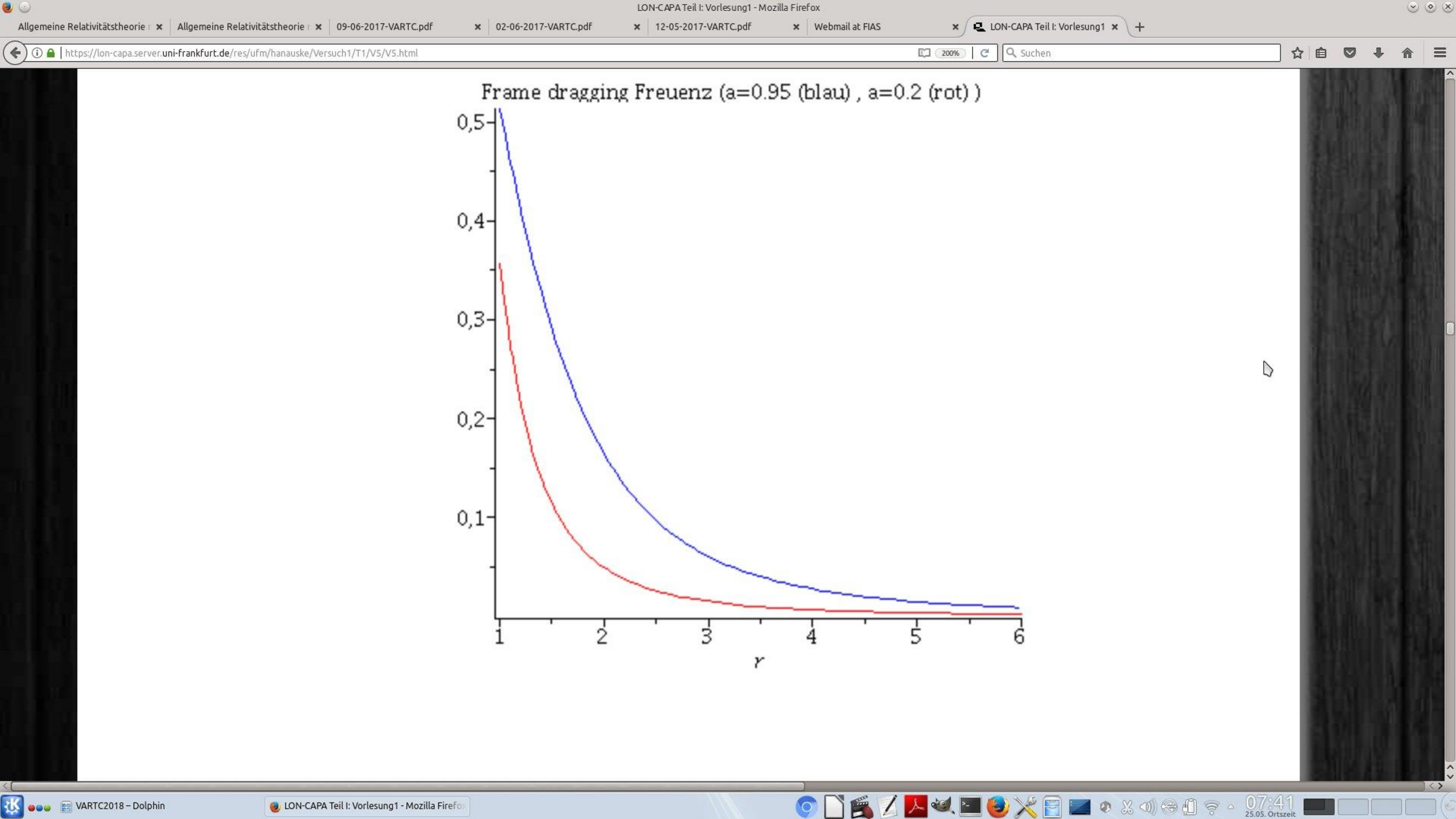


Die Rotation der raumzeitlichen Struktur um das schwarze Loch (das Frame dragging)

Ein rotierendes schwarzes Loch zieht die Raumzeit mit sich mit. Die Rotationsfrequenz mit der die Raumzeit mitgeführt wird nennt man "Frame dragging" Frequenz; sie quantisiert, mit welcher Frequenz ein im Eigensystem "ruhender" Probekörper von der ihm zugrundeliegenden Raumzeit mitgezogen wird:

$$\omega(r) = \frac{d\phi}{dt} = \frac{\frac{d\phi}{d\tau}}{\frac{dt}{d\tau}} = \frac{u^\phi}{u^t} = \frac{g^{t\phi}}{g^{tt}}$$

```
> FrameDrag:=get_compts(ginv)[1,4]/get_compts(ginv)[1,1];
   #FrameDrag:=-get_compts(g)[1,4]/get_compts(g)[4,4];
```


LON-CAPA Teil I: Vorlesung1 - Mozilla Firefox
Allgemeine Relativitätstheorie
Allgemeine Relativitätstheorie
09-06-2017-VARTC.pdf
02-06-2017-VARTC.pdf
12-05-2017-VARTC.pdf
Webmail at FIAS
LON-CAPA Teil I: Vorlesung1
https://lon-capa.server.uni-frankfurt.de/res/ufm/hanauske/Versuch1/T1/V5/V5.html
Suchen
Frame dragging Freuenz (a=0.95 (blau) , a=0.2 (rot))
0,5
0,4
0,3
0,2
0,1
1
2
3
4
5
6
r
VARTC2018 – Dolphin
LON-CAPA Teil I: Vorlesung1 - Mozilla Firefox
07:41

LON-CAPA Teil I: Vorlesung1 - Mozilla Firefox
Allgemeine Relativitätstheorie | Allgemeine Relativitätstheorie | 09-06-2017-VARTC.pdf | 02-06-2017-VARTC.pdf | 12-05-2017-VARTC.pdf | Webmail at FIAS | LON-CAPA Teil I: Vorlesung1
https://lon-capa.server.uni-frankfurt.de/res/ufm/hanauske/Versuch1/T1/V5/V5.html

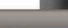
Radial in ein rotierendes schwarzes Loch einfallender Probekörper

Wir betrachten nun einen Probekörper der radial in ein rotierendes schwarzes Loch fällt.

```
>    restart:
 with( tensor ):
   with(plots):
 with(plottools):
```

Definition der kovarianten Raumzeit-Metrik eines rotierenden schwarzen Lochs der Masse M und Rotation a in Kerrschildkoordinaten:

```
>                 coord := [t, r, theta, phi]:
                  rho2:=r^2+(a*cos(theta))^2:
 Delta:=r^2-2*M*r+a^2:Sig2:=(r^2+a^2)^2-a^2*Delta*(sin(theta))^2:
         g_compts := array(symmetric,sparse, 1..4, 1..4):
                g_compts[1,1] := (1-2*M*r/rho2):
         g_compts[1,4] := +(2*a*M*r*(sin(theta))^2)/rho2:
                  g_compts[2,2] :=-rho2/Delta:
                    g_compts[3,3] := -rho2:
```

VARTC2018 – Dolphin | LON-CAPA Teil I: Vorlesung1 - Mozilla Firefox

07:41
25.05. Ortszeit

LON-CAPA Teil I: Vorlesung1 - Mozilla Firefox
https://lon-capa.server.uni-frankfurt.de/res/ufm/hanauske/Versuch1/T1/V5/V5.html

Anfangswerte:

Zur Zeit t=0 sei der fallende Körper an der folgenden Position: (r=10,theta=Pi/2,phi=0), die Anfangsgeschwindigkeit des Körpers sei 0. Wir beschreiben den Fall aus der Sichtweise eines im Unendlichen ruhenden Beobachters.Bemerkung: Der Anfangswert dt0 ergibt sich hierbei aus dem infinitesimalen Weglängenelements $ds^2=1$ eines massiven Probekörpers:

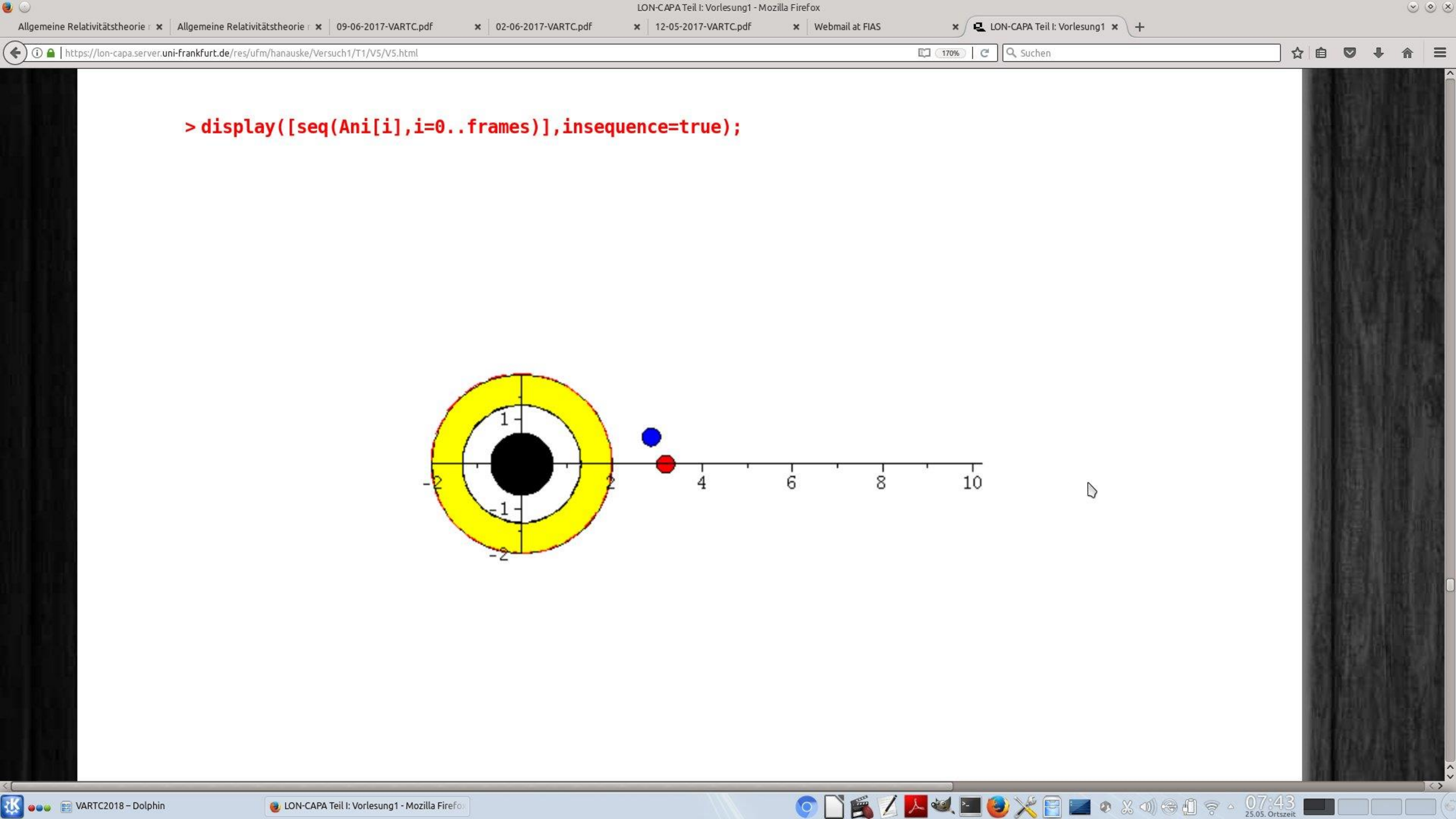
```
> dt=solve(evalf(subs({M=setM,a=seta,theta=Pi/2,dr=0,dtheta=0,dphi=0,r=10},ds2=1)),dt);
```

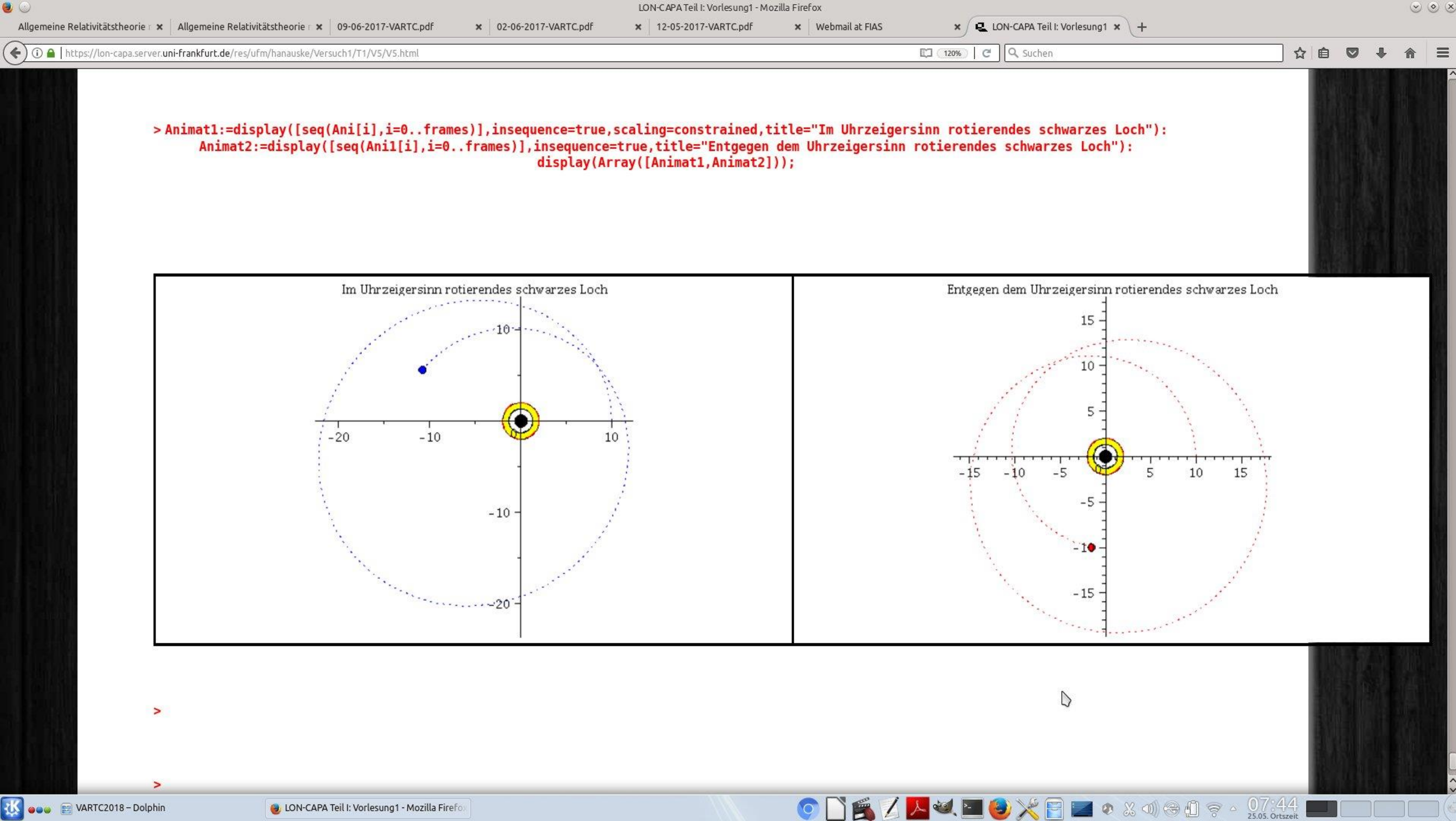
$$dt = (1.118033989, -1.118033989) \quad (2.2.4)$$

```
>           r0:=10:
            t0:=0:
         theta0:=Pi/2:
           phi0:=0:
            dr0:=0:
 dt0:=evalf(1/sqrt(1-2/r0)):
          dtheta0:=0:
           dphi0:=0:
```

Numerisches Lösen der Geodätengleichung:

VARTC2018 – Dolphin   LON-CAPA Teil I: Vorlesung1 - Mozilla Firefox   07:42 25.05.


LON-CAPA Teil I: Vorlesung1 - Mozilla Firefox
https://lon-capa.server.uni-frankfurt.de/res/ufm/hanauske/Versuch1/T1/V5/V5.html
> display([seq(Ani[i],i=0..frames)],insequence=true);
1
-1
-2
-2
2
4
6
8
10



> Animat1:=display([seq(Ani[i],i=0..frames)],insequence=true,scaling=constrained,title="Im Uhrzeigersinn rotierendes schwarzes Loch"):
Animat2:=display([seq(Ani1[i],i=0..frames)],insequence=true,title="Entgegen dem Uhrzeigersinn rotierendes schwarzes Loch"):
display(Array([Animat1,Animat2]));
Im Uhrzeigersinn rotierendes schwarzes Loch
Entgegen dem Uhrzeigersinn rotierendes schwarzes Loch

LON-CAPA Teil I: Vorlesung1 - Mozilla Firefox
Allgemeine Relativitätstheorie | Allgemeine Relativitätstheorie | 09-06-2017-VARTC.pdf | 02-06-2017-VARTC.pdf | 12-05-2017-VARTC.pdf | Webmail at FIAS | LON-CAPA Teil I: Vorlesung1
https://lon-capa.server.uni-frankfurt.de/res/ufm/hanauske/Versuch1/T1/V6/V6.html

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 2016)

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

Frankfurt am Main 11.04.2016

Erster Vorlesungsteil: Allgemeine Relativitätstheorie mit Maple

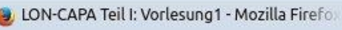
6. Vorlesung

Die Kerr Metrik: Effektives Potential, kreisförmige Bewegungen, die innerste stabile Kreisbahn und der gravitomagnetische Effekt

Basierend auf den Ergebnissen der geodätischen Bewegung eines Probekörpers um ein rotierendes Kerr schwarzes Loch (siehe Vorlesung 5), werden die möglichen Bewegungen mittels eines definierten effektiven Potential näher verstanden. Zusätzlich wird der durch den Mitführungseffekt der Raumzeit ("Frame-Dragging" bzw. Lense-Thiring Effekt) verursachte, gravitomagnetische Effekt an einem speziellen Beispiel veranschaulicht.

Bahnbewegungen in der Ebene und das effektive Potential V(r)

ähnlich wie im nichtrotierenden Fall (siehe Vorlesung 3) charakterisieren wir die unterschiedlichen Bahnbewegungen mittels eines effektiven Potentials:


VARTC2018 – Dolphin
LON-CAPA Teil I: Vorlesung1 - Mozilla Firefox

07:46
25.05. Ortszeit

Berechnung der Geodätengleichung als Funktion des affinen Parameters λ: Die Geodätengleichung ist ein System gekoppelter Differentialgleichungen

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

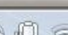
wobei λ ein affiner Parameter (z.B. die Eigenzeit), t, r, θ und ϕ die sphärischen Koordinaten und $\Gamma^\mu_{\nu\rho}$ die Christoffel Symbole zweiter Art darstellen.

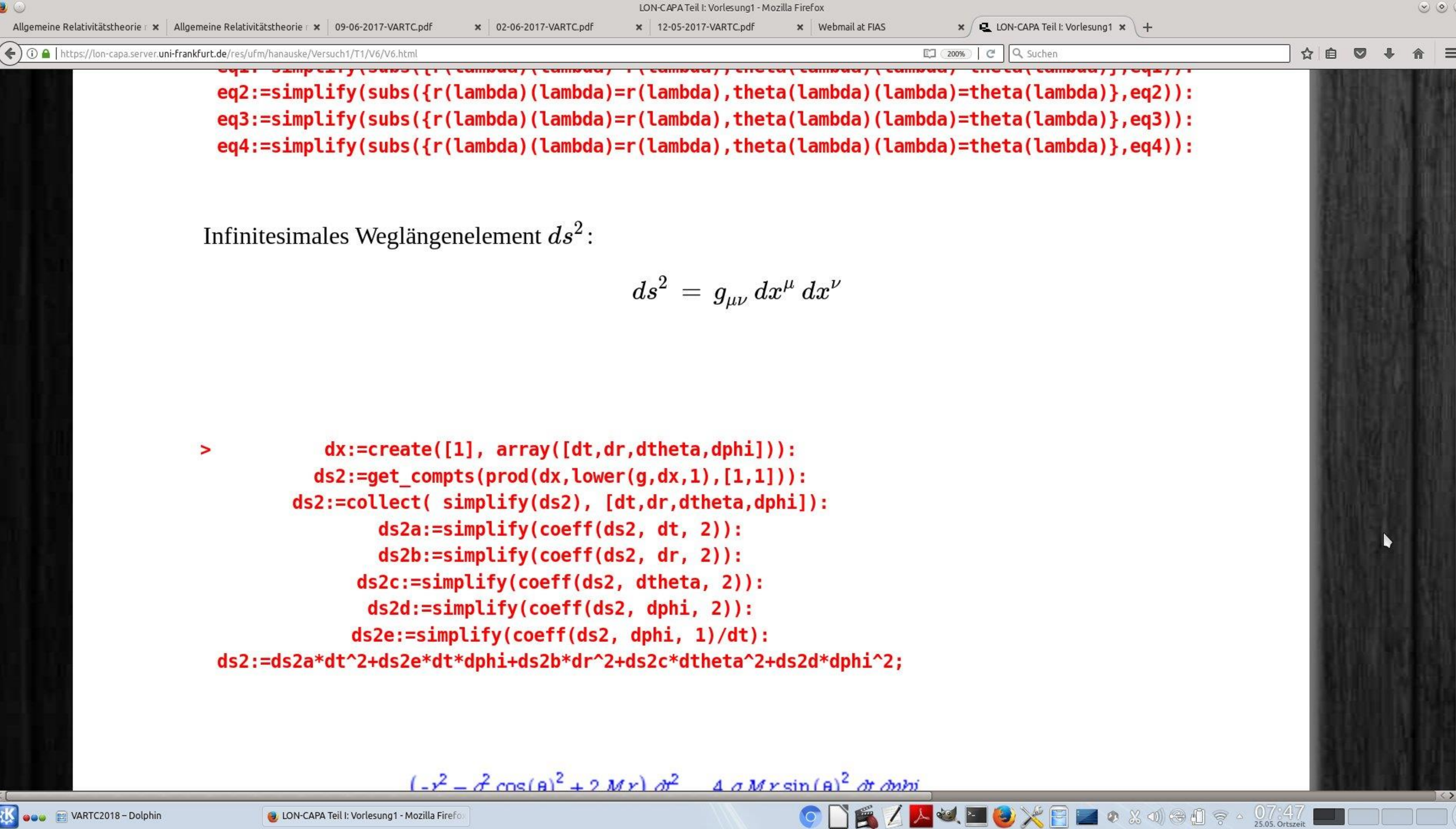
```
> eqns:=geodesic_eqns( coord, lambda, Cf2 );
```

$$eqns := \left[\frac{d^2}{d\lambda^2} \phi(\lambda) - \left(2 \sin(\theta)^2 M a \left(-r^2 + a^2 \cos(\theta)^2 \right) \left(\frac{d}{d\lambda} t(\lambda) \right) \left(\frac{d}{d\lambda} r(\lambda) \right) \right) \Big/ \right.$$

$$\left(a^6 \cos(\theta)^4 + a^4 \cos(\theta)^4 r^2 + 2 r^2 a^4 \cos(\theta)^2 + 2 r^4 a^2 \cos(\theta)^2 \right.$$

$$- 4 M r^3 a^2 \cos(\theta)^2 + r^6 + r^4 a^2 - 2 M r^5 - 4 M^2 r^2 a^2 \cos(\theta)^2 - 2 \cos(\theta)^4 M r a^4$$




LON-CAPA Teil I: Vorlesung1 - Mozilla Firefox
https://lon-capa.server.uni-frankfurt.de/res/ufm/hanauske/Versuch1/T1/V6/V6.html
eq2:=simplify(subs({r(lambda)(lambda)=r(lambda),theta(lambda)(lambda)=theta(lambda)},eq2)):
eq3:=simplify(subs({r(lambda)(lambda)=r(lambda),theta(lambda)(lambda)=theta(lambda)},eq3)):
eq4:=simplify(subs({r(lambda)(lambda)=r(lambda),theta(lambda)(lambda)=theta(lambda)},eq4)):
Infinitesimales Weglängenelement ds^2:
$ds^2 = g_{\mu\nu} \, dx^\mu \, dx^\nu$
> dx:=create([1], array([dt,dr,dtheta,dphi])):
ds2:=get_compts(prod(dx,lower(g,dx,1),[1,1])):
ds2:=collect(simplify(ds2), [dt,dr,dtheta,dphi]):
ds2a:=simplify(coeff(ds2, dt, 2)):
ds2b:=simplify(coeff(ds2, dr, 2)):
ds2c:=simplify(coeff(ds2, dtheta, 2)):
ds2d:=simplify(coeff(ds2, dphi, 2)):
ds2e:=simplify(coeff(ds2, dphi, 1)/dt):
ds2:=ds2a*dt^2+ds2e*dt*dphi+ds2b*dr^2+ds2c*dtheta^2+ds2d*dphi^2;

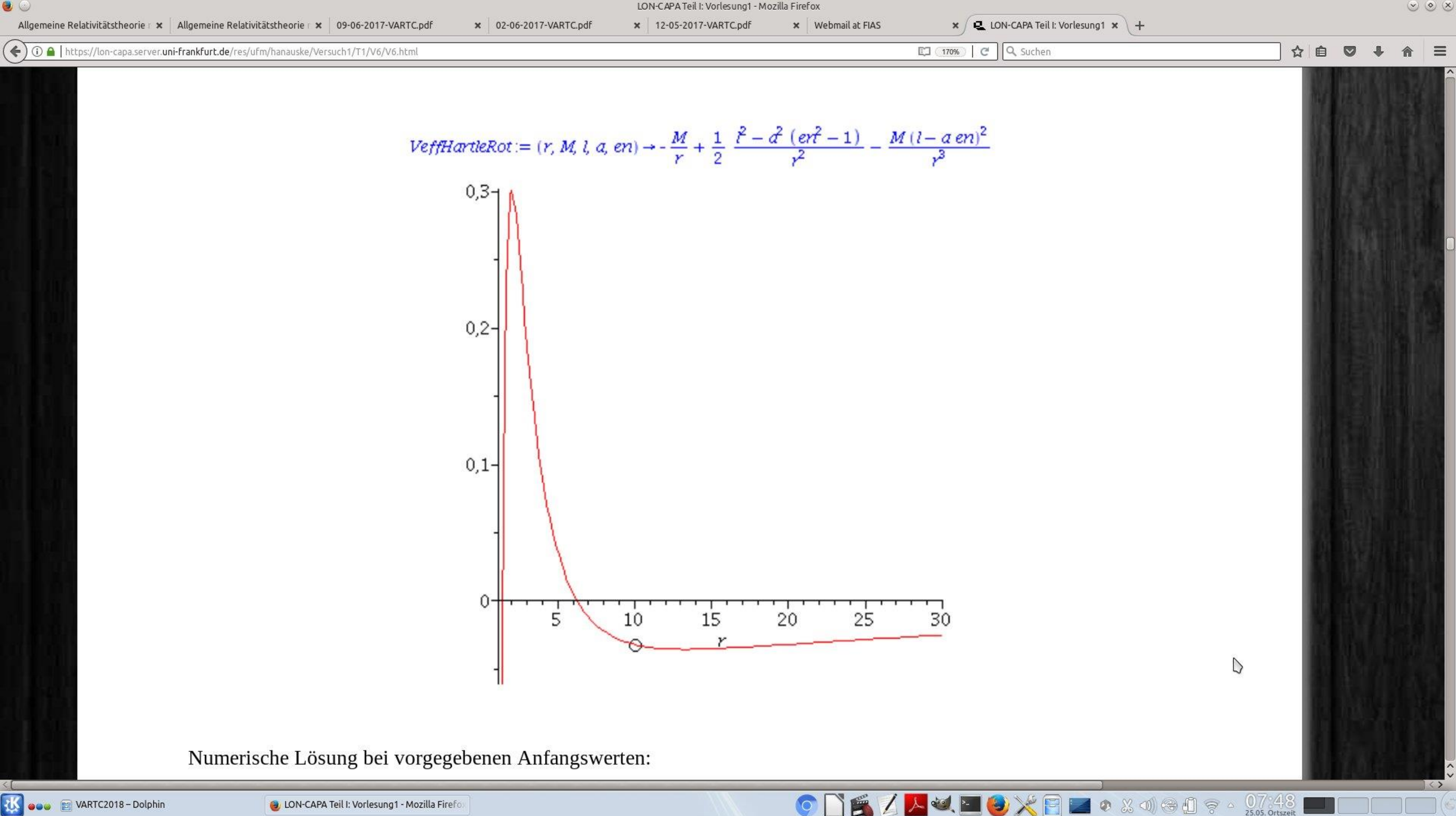
LON-CAPA Teil I: Vorlesung1 - Mozilla Firefox
https://lon-capa.server.uni-frankfurt.de/res/ufm/hanauske/Versuch1/T1/V6/V6.html

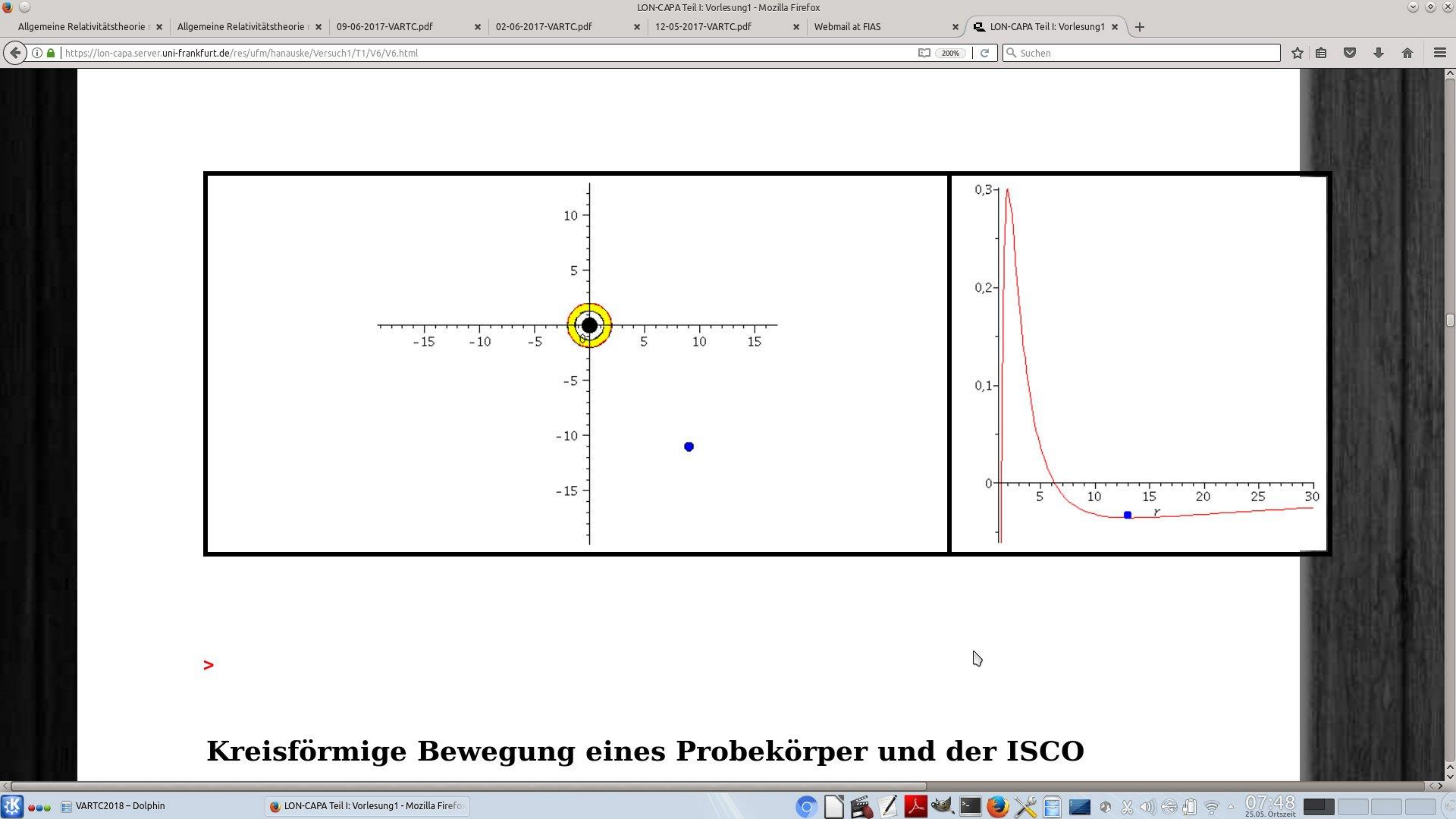
Festlegung der Anfangswerte:

```
> setM:=1:
  seta:=0.95:
  r0:=10:
  t0:=0:
  theta0:=Pi/2:
  phi0:=0:
  dr0:=0:
  dtheta0:=0:
  dphi0:=0.041:
  dt0:=solve(subs({theta=theta0,dphi=dphi0,dr=dr0,dtheta=dtheta0,M=1,a=0.95,r=10},ds2)=1,dt)[1]:
```

In der Literatur wird die Bewegung eines Probekörpers um ein rotierendes schwarzes Loch mittels eines definierten, effektiven Potentials illustriert (siehe z.B. Hartle- bzw. Hobson Buch). Dieses Potential hängt von dem, bei der Bewegung erhaltenem Drehimpuls pro Masse m und der Probekörper-Energie pro Masse ab. Die im Zentralfeld möglichen Bewegungen werden mittels zweier erhaltener Größen (l: Drehimpuls pro Masse m und E: Energie pro Masse) charakteriesiert. Die folgende Abbildung zeigt (in der Nomenklatur vom Hartle-Buch) das effektive Potential als Funktion des Radius bei den obigen gewählten Anfangswerten:

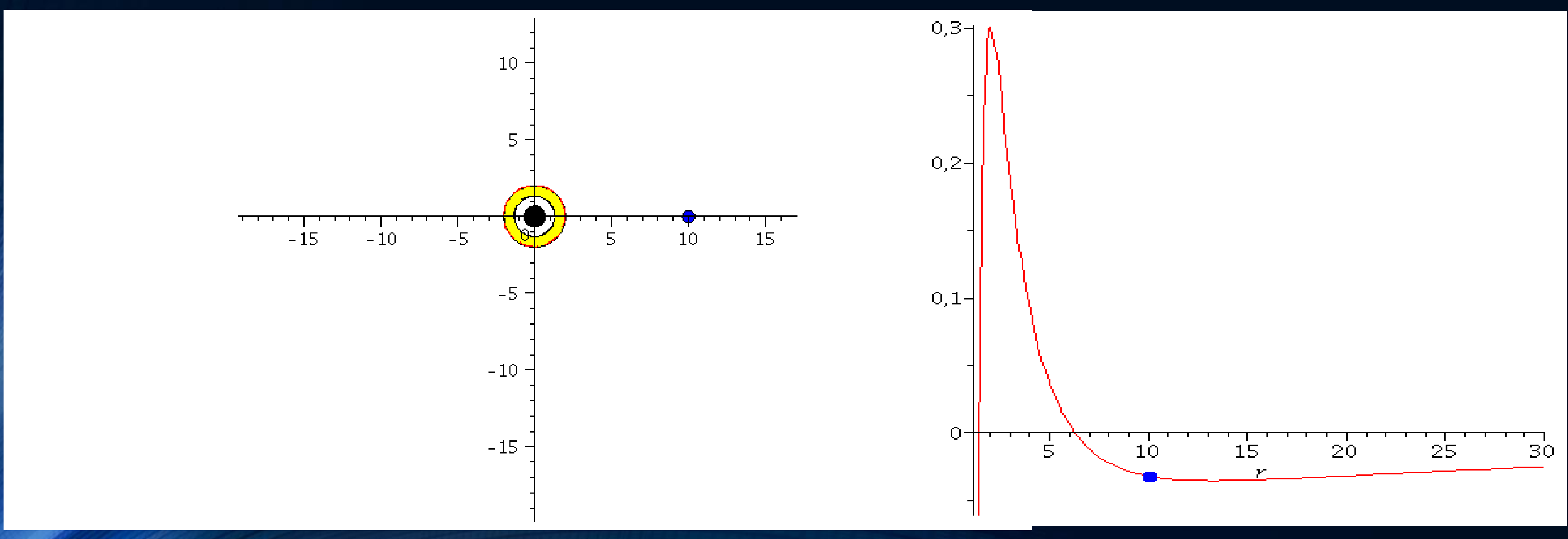
$$V_{eff}(r, M, l, a, E) = -\frac{M}{r} + \frac{l^2 - a^2\left(E^2 - 1\right)}{2\,r^2} - \frac{M(l - a\,E)^2}{r^3}$$


LON-CAPA Teil I: Vorlesung1 - Mozilla Firefox
https://lon-capa.server.uni-frankfurt.de/res/ufm/hanauske/Versuch1/T1/V6/V6.html
VeffHartleRot := (r, M, l, a, en) → - M/r + 1/2 (l^2 - a^2 (en^2 - 1))/r^2 - M (l - a en)^2/r^3
0,3
0,2
0,1
0
5
10
15
20
25
30
r
Numerische Lösung bei vorgegebenen Anfangswerten:


LON-CAPA Teil I: Vorlesung1 - Mozilla Firefox
Allgemeine Relativitätstheorie
Allgemeine Relativitätstheorie
09-06-2017-VARTC.pdf
02-06-2017-VARTC.pdf
12-05-2017-VARTC.pdf
Webmail at FIAS
LON-CAPA Teil I: Vorlesung1
https://lon-capa.server.uni-frankfurt.de/res/ufm/hanauske/Versuch1/T1/V6/V6.html
Suchen
10
5
-5
-10
-15
-15
-10
-5
5
10
15
0,3
0,2
0,1
0
5
10
15
20
25
30
r
>
Kreisförmige Bewegung eines Probekörper und der ISCO
VARTC2018 – Dolphin
LON-CAPA Teil I: Vorlesung1 - Mozilla Firefox
07:48
25.05. Ortszeit

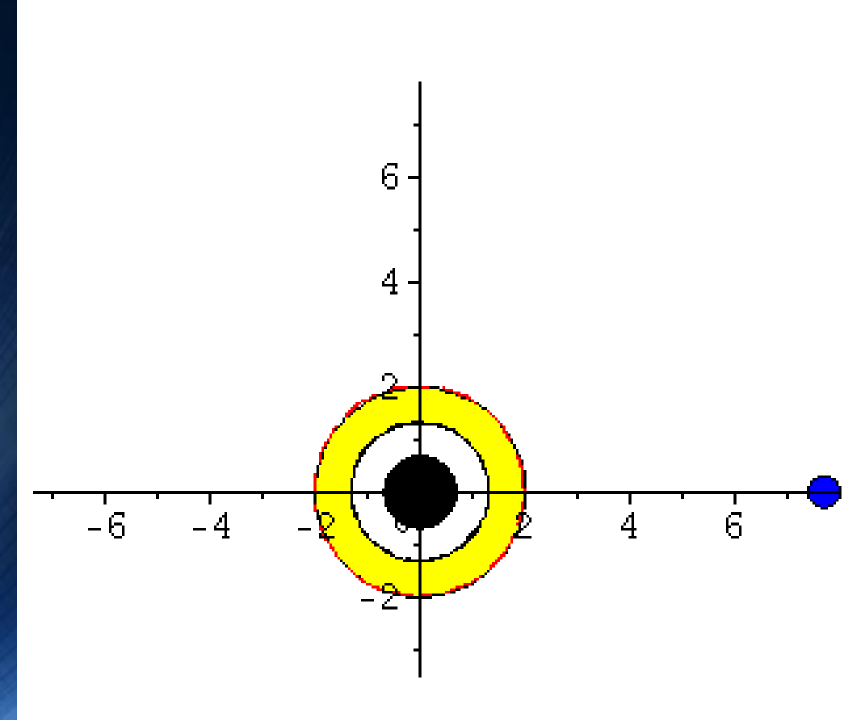
Kerr Metrik: Effektives Potential

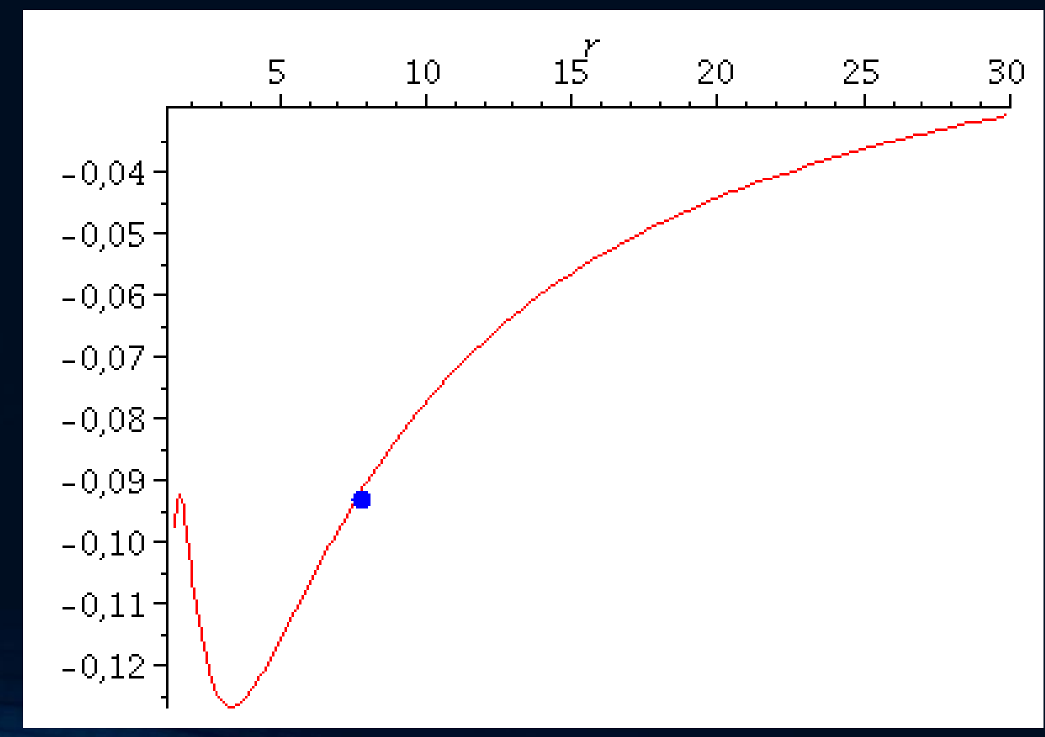
$$V_{eff}(r, M, l, a, E) = -\frac{M}{r} + \frac{l^2 - a^2\left(E^2 - 1\right)}{2\,r^2} - \frac{M(l - a\,E)^2}{r^3}$$



Kerr Metrik: Effektives Potential

$$V_{eff}(r, M, l, a, E) = -\frac{M}{r} + \frac{l^2 - a^2\left(E^2 - 1\right)}{2\,r^2} - \frac{M(l - a\,E)^2}{r^3}$$

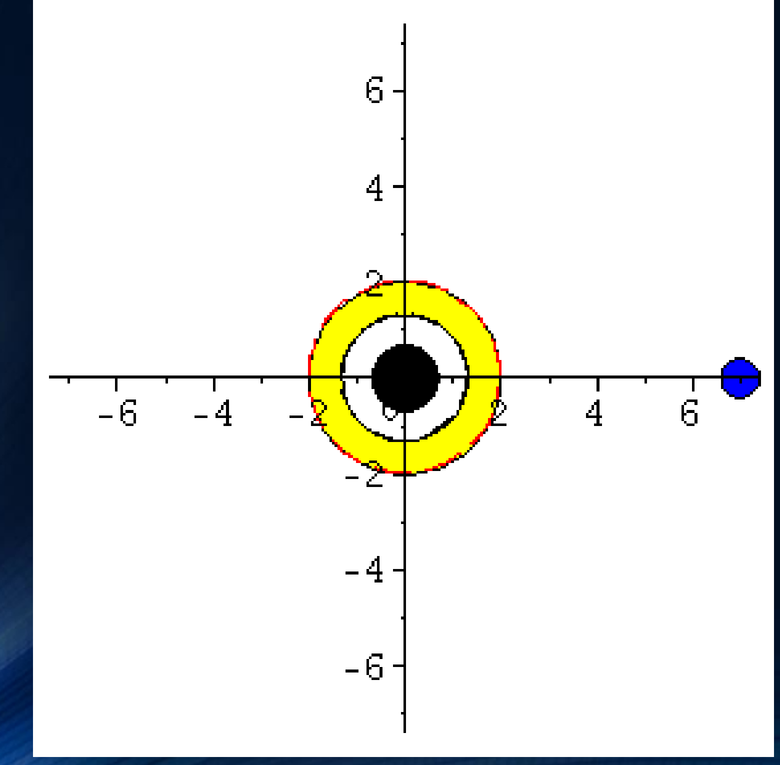


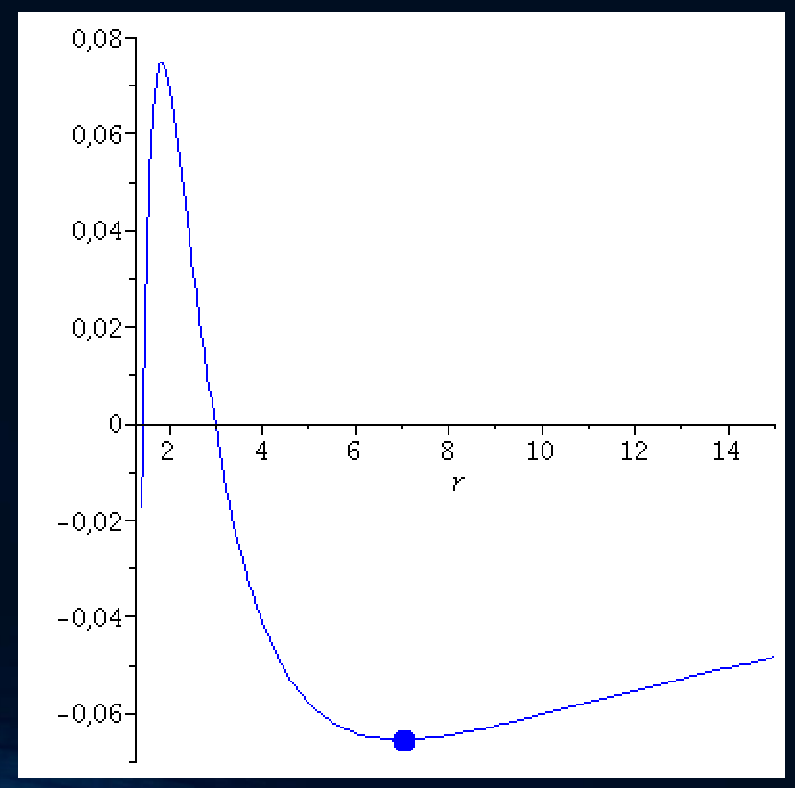


Kerr Metrik: Effektives Potential

Kreisförmige Bahnbewegungen

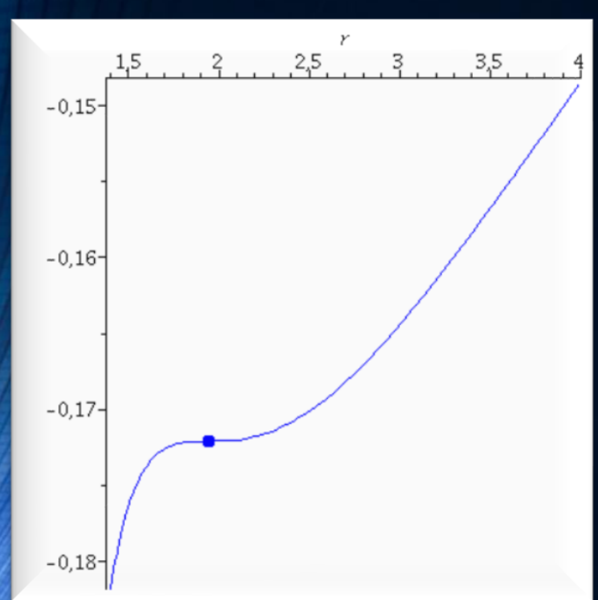
$$V_{eff}(r, M, l, a, E) = -\frac{M}{r} + \frac{l^2 - a^2\left(E^2 - 1\right)}{2\,r^2} - \frac{M(l - a\,E)^2}{r^3}$$



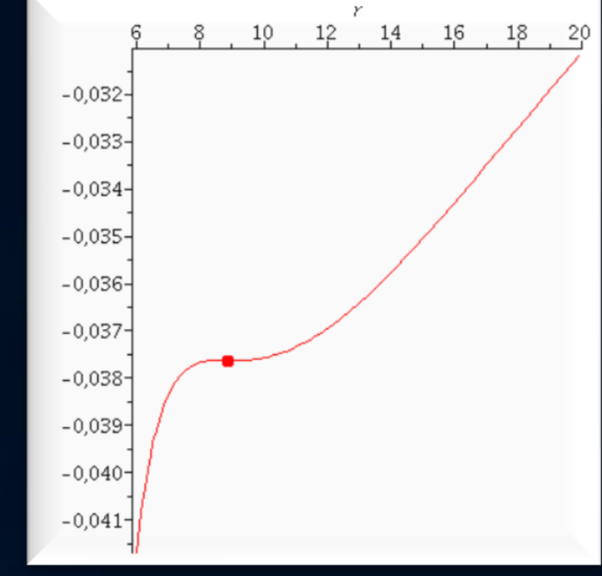


Kerr Metrik: Effektives Potential
Innerste „stabile“ kreisförmige Bahnbewegungen (ISCOs)

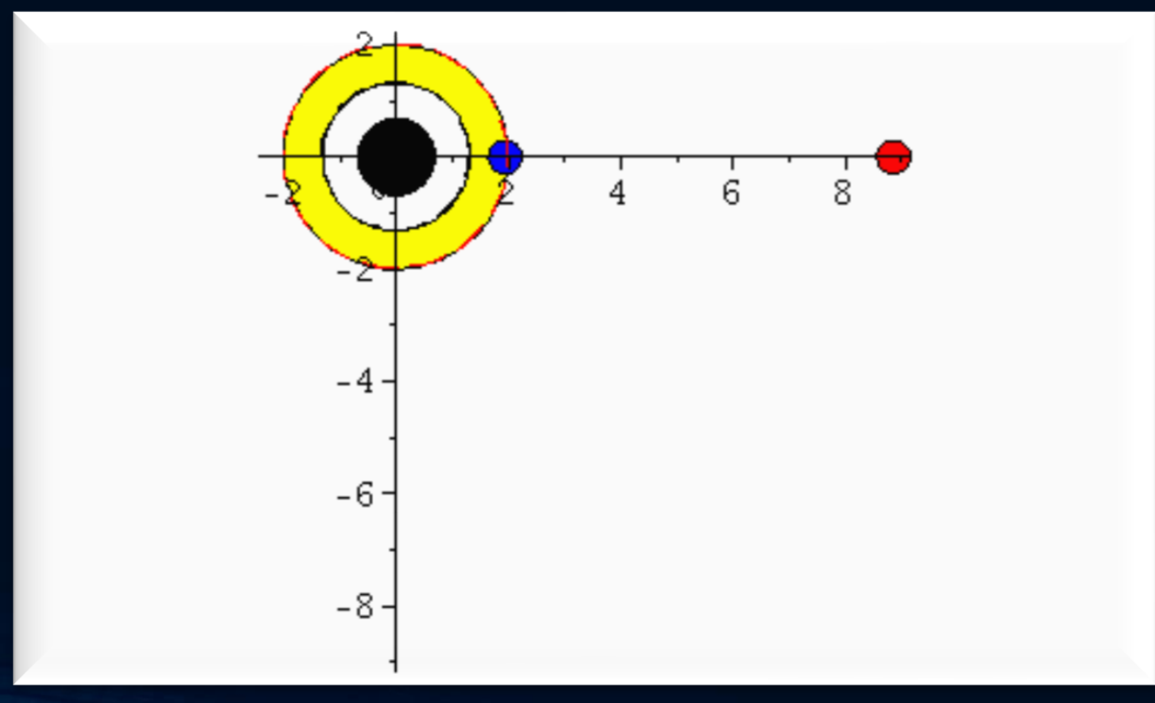
$$V_{eff}(r, M, l, a, E) = -\frac{M}{r} + \frac{l^2 - a^2\left(E^2 - 1\right)}{2\,r^2} - \frac{M(l - a\,E)^2}{r^3}$$



Probekörper rotiert mit der Rotationsrichtung des schwarzen Loches

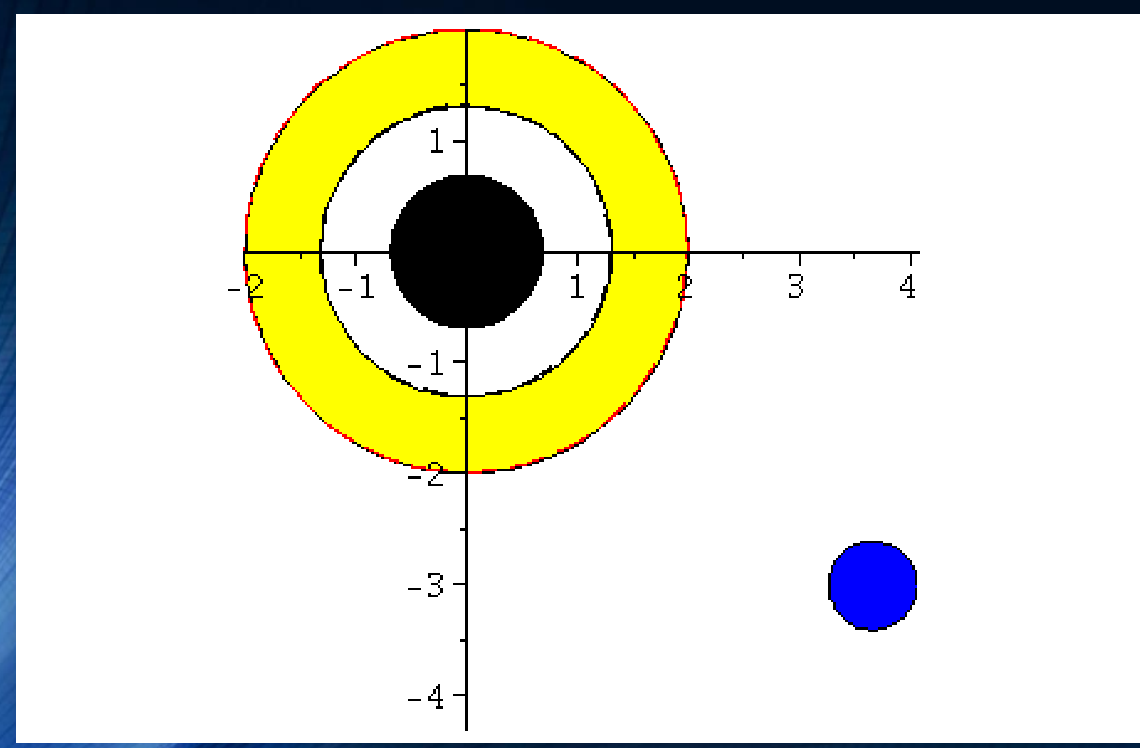


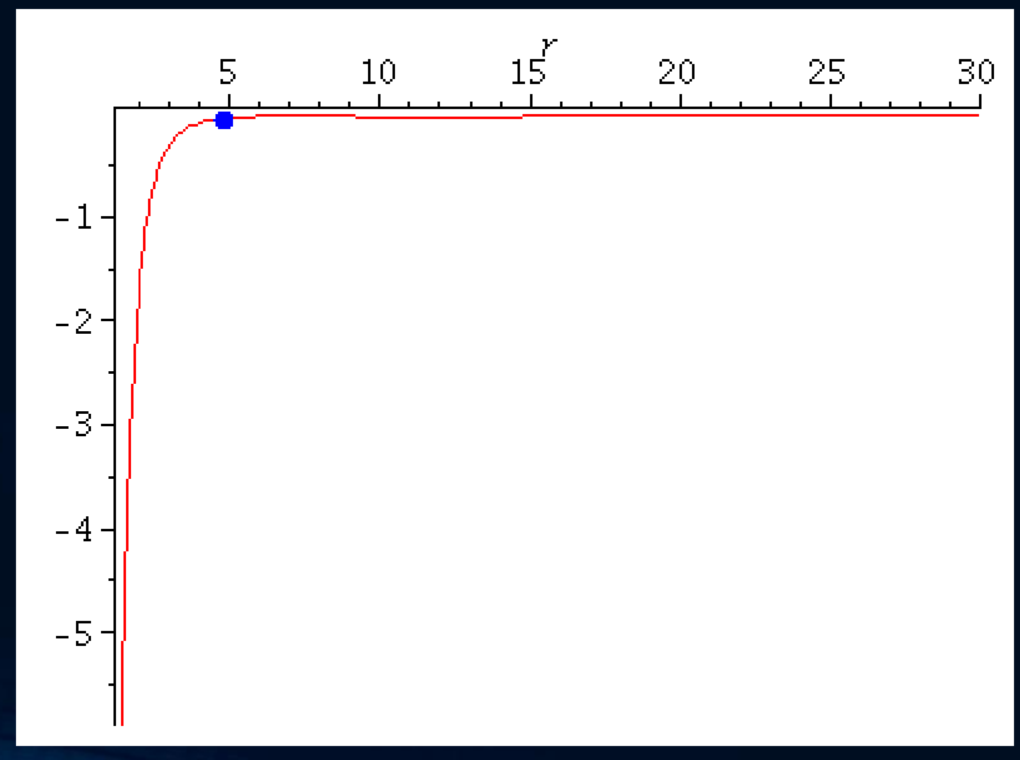
Probekörper rotiert entgegen der Rotationsrichtung des schwarzen Loches



Kerr Metrik: Bewegungen innerhalb der Ergosphäre

$$V_{eff}(r, M, l, a, E) = -\frac{M}{r} + \frac{l^2 - a^2\left(E^2 - 1\right)}{2\,r^2} - \frac{M(l - a\,E)^2}{r^3}$$





Der gravitomagnetische Effekt

Beobachtung

a) + b)

+ −

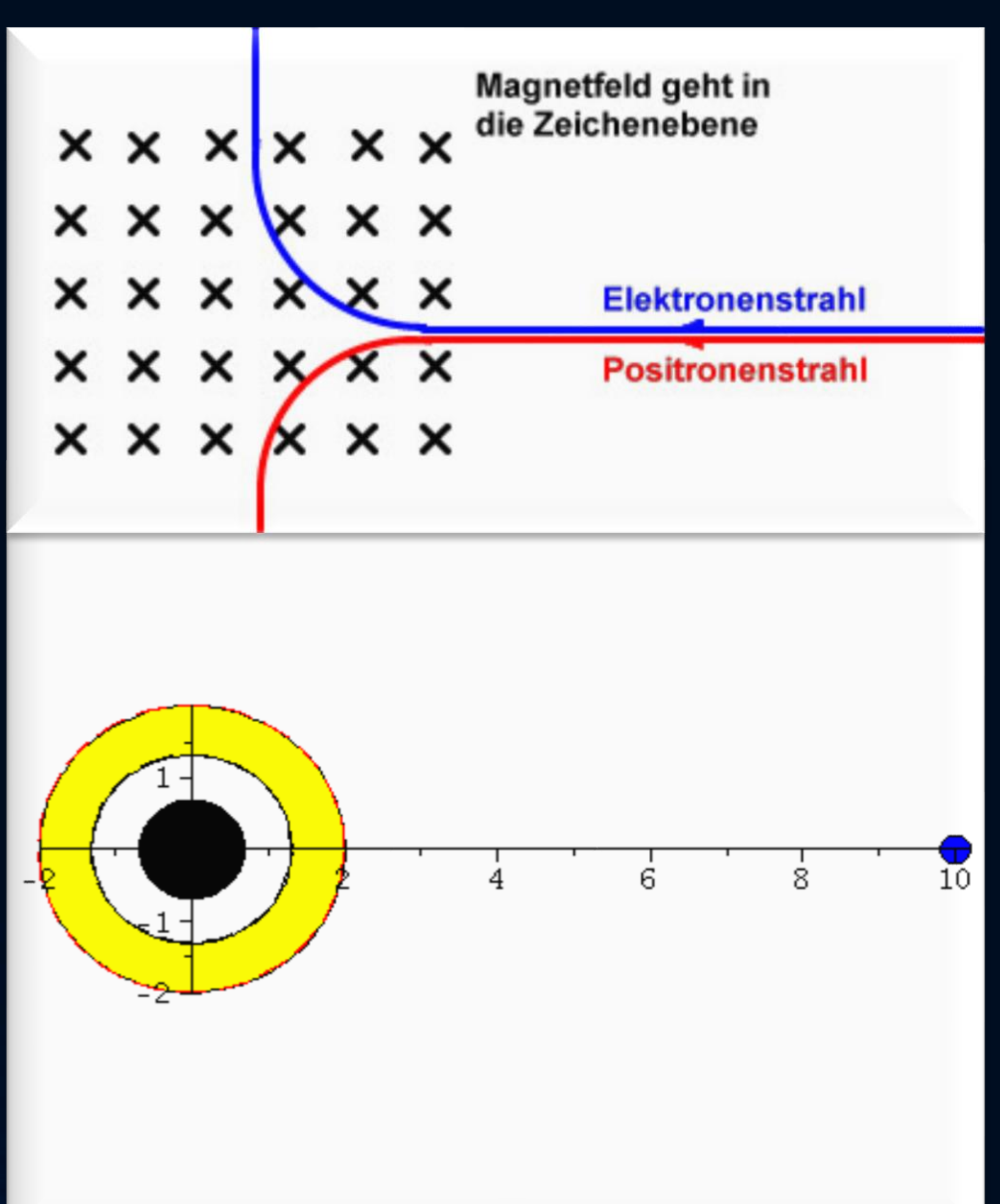
− +

Versuche eine Regel mit Daumen, Zeigefinger und Mittelfinger deiner linken Hand zu formulieren.

+ −

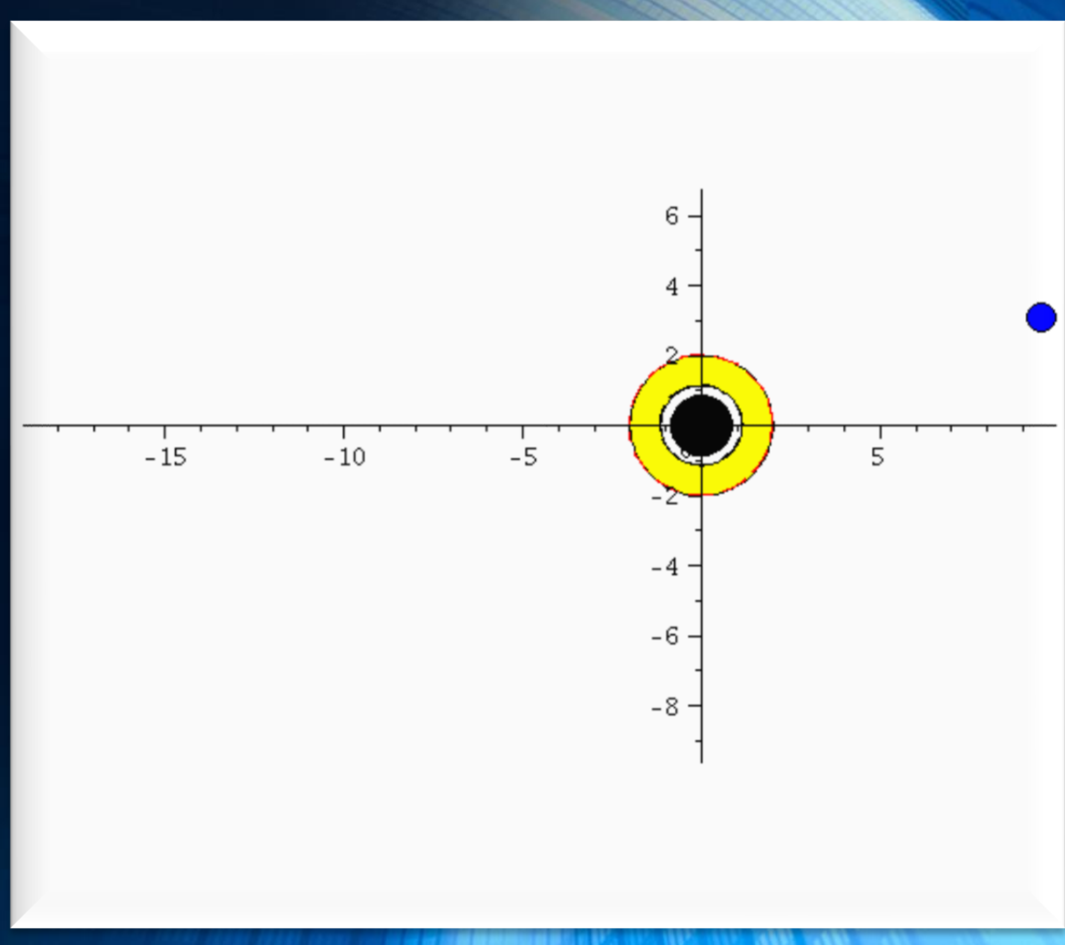
− +

(C) Lorenz K Schröfl



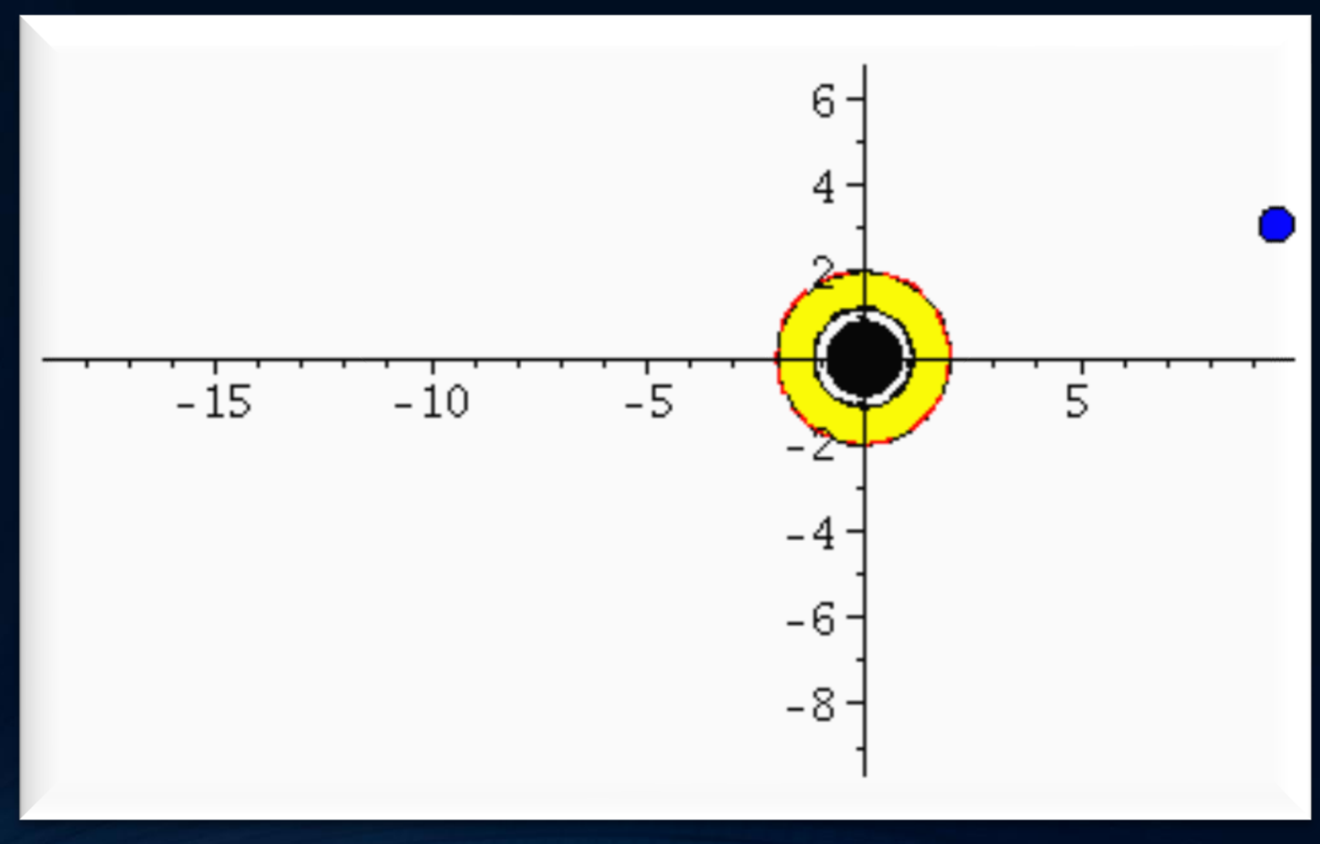
Der gravitomagnetische Effekt

r
$F_{Lorentz}$
v_0
$F_{Lorentz}$
e^-
e^-
v_0



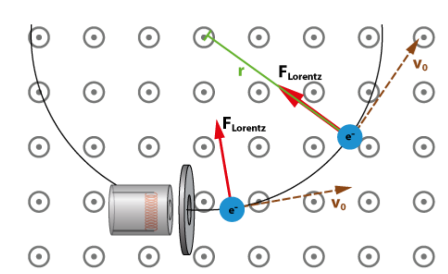
Kerr Metrik: Der gravitomagnetische Effekt

Die grüne Kurve entspricht einer Situation ohne Magnetfeld (nur Coulombkraft = nur gravitative Anziehung, keine Rotation), die blaue Kurve entspricht einer Situation wo das gravitomagnetische Feld in +z-Richtung (schwarzes Loch rotiert entgegen dem Uhrzeigersinn) zeigt und bei der roten Kurve zeigt das gravitomagnetische Feld in -z-Richtung (schwarzes Loch rotiert im Uhrzeigersinn).



Der gravitomagnetische Effekt

Elektromagnetischer Effekt der Lorentzkraft:



Gravito-magnetischer Effekt:

$$\omega(r,\theta) = \frac{d\phi}{dt} = \frac{\frac{d\phi}{d\tau}}{\frac{dt}{d\tau}} = \frac{u^\phi}{u^t} = \frac{g^{t\phi}}{g^{tt}}$$

Diese "Frame dragging" Frequenz wirkt in ähnlicher Weise auf die Geschwindigkeit von Probekörpern, wie das Magnetfeld in der Elektrodynamik die Lorentzkraft verursacht. Im Fall1 (grün) ist das gravitomagnetische Feld Null, im Fall 2 (blau) ist es aus der äquatoriellen Ebene nach oben gerichtet und im Fall 3 zeigt es nach unten. In erster Näherung (siehe Fließbach Buch, S:172) ist die gravitomagnetische Lorentzkraft gleich $\sim 2\,(\omega \times \mathbf{v})$, wobei $\times$ das Kreuzprodukt, ω der axiale Vektor der "Frame dragging" Frequenz und $\mathbf{v}$ der Geschwindigkeitsvektor des Probekörpers ist. Die Änderung des Geschwindigkeitsvektors nimmt in Schwachfeldnäherung dann die folgende Gestalt an:

$$\frac{d\mathbf{v}}{d\tau} = \underbrace{-\text{grad}\,\Phi(\mathbf{r})}_{\text{gewoehnlicher Teil der gravitativen Kraft}} + \underbrace{2\,\omega(\mathbf{r}) \times \mathbf{v}}_{\text{gravitomagnetische Lorentzkraft}} + \mathcal{O}(v^2/c^2)\;,$$

wobei $\Phi(\mathbf{r})$ das Newtonsche Gravitationspotential und $\mathbf{v} = (v^r, v^\theta, v^\phi)$ der Geschwindigkeitsvektor des Probekörpers ist. Die unten abgebildete Grafik zeigt die "Frame dragging" Frequenz $\omega = \omega_z(r)$ für die Kerr Metrik, wobei bei der schwarzen Kurve a=0, bei der blauen Kurve a=0.99 und bei der roten Kurve a=-0.99 ist.

Beispiel: Das schwarze Loch in M87

Echte Singularität im Zentrum

Ereignishorizont

Photonensphäre
Letzte stabile kreisförmige Bahnbewegung eines masselosen Teilchens

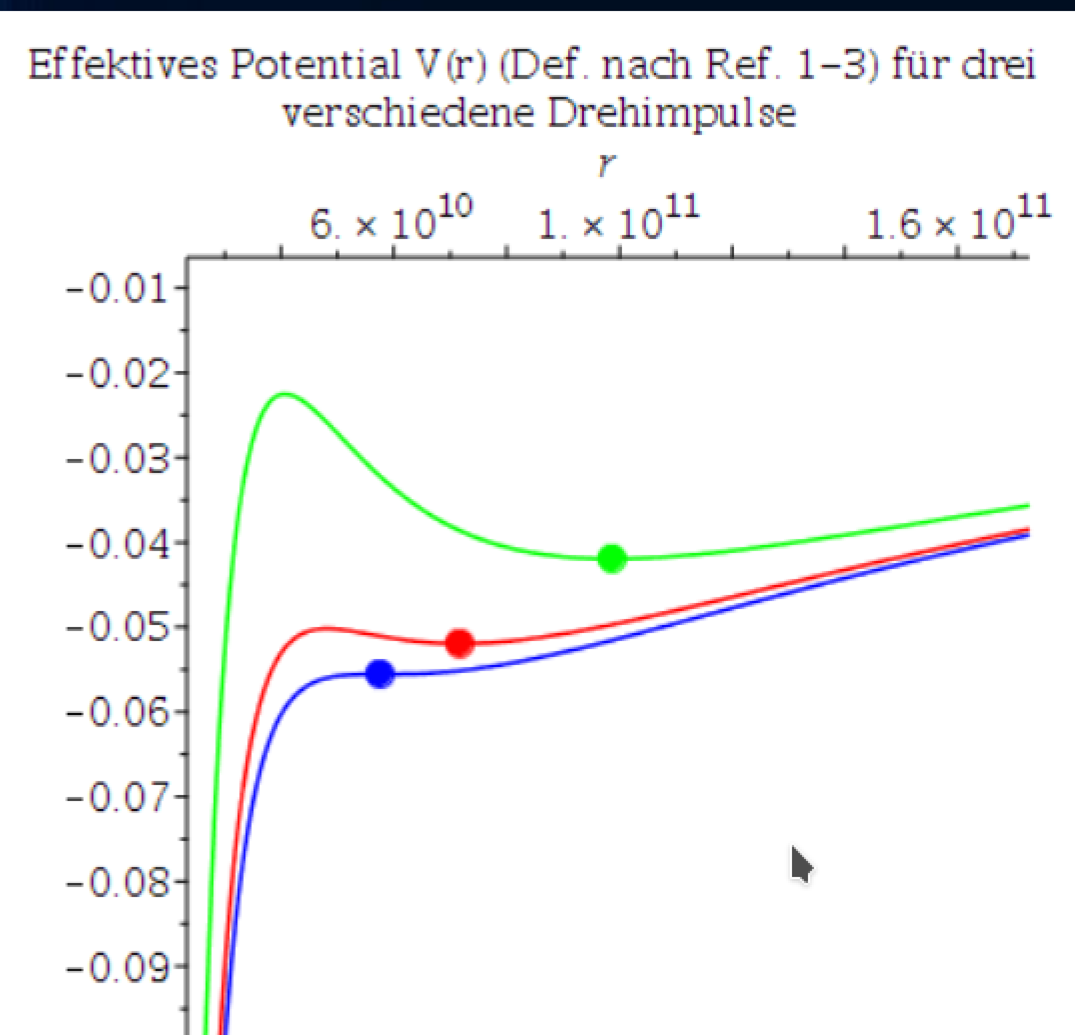
Akkretionsscheibe
Letzte stabile kreisförmige Bahnbewegung eines massiven Probekörpers
Last Stable Circular Orbit (ISCO)

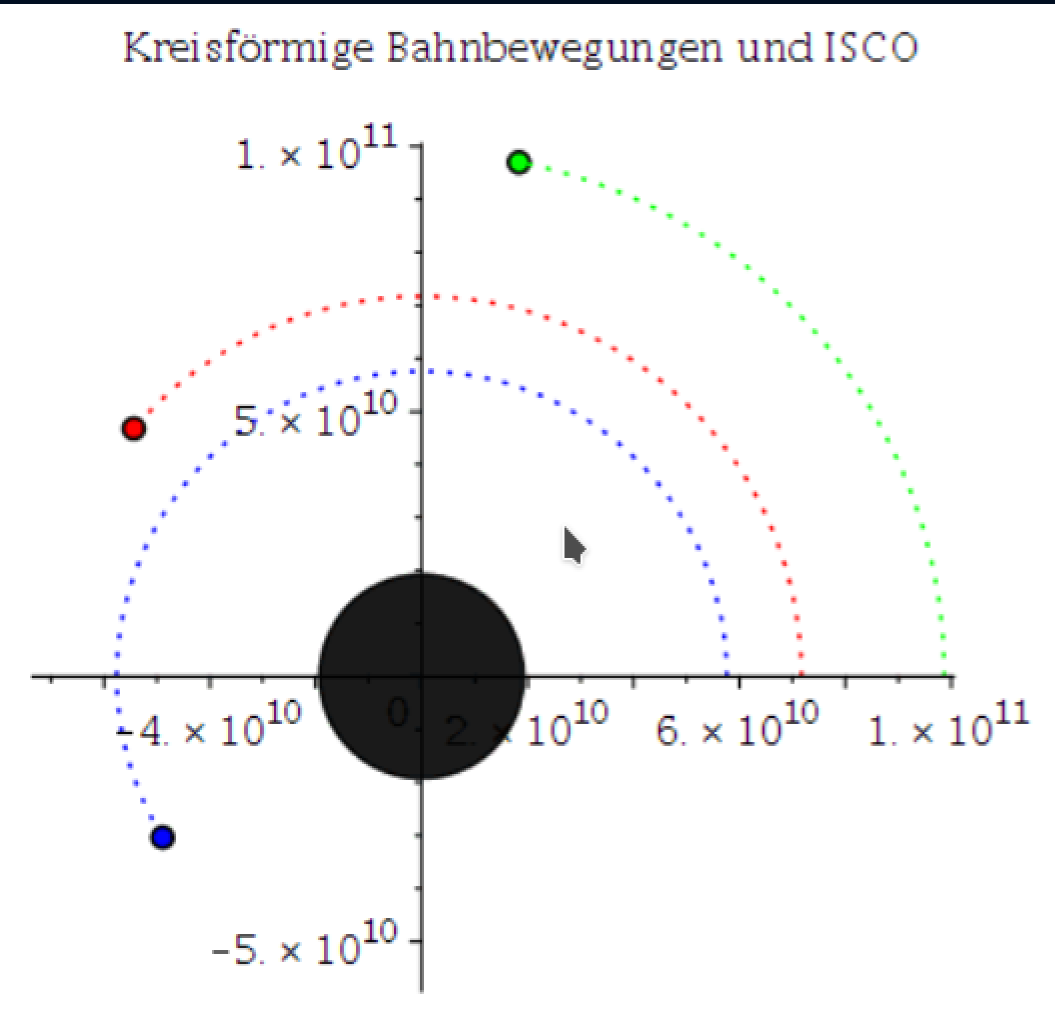
Ein wenig mehr als hundert Jahre nachdem Albert Einstein seine Feldgleichungen der *Allgemeine Relativiätstheorie* der Öffentlichkeit präsentierte, und er damit die Grundlage für Gravitationswellen und schwarzer Löcher formulierte, ist seit einigen Wochen ein Meilenstein in der Geschichte der Astronomie in aller Munde (erstes Bild eines schwarzen Lochs, siehe rechte Abbildung).

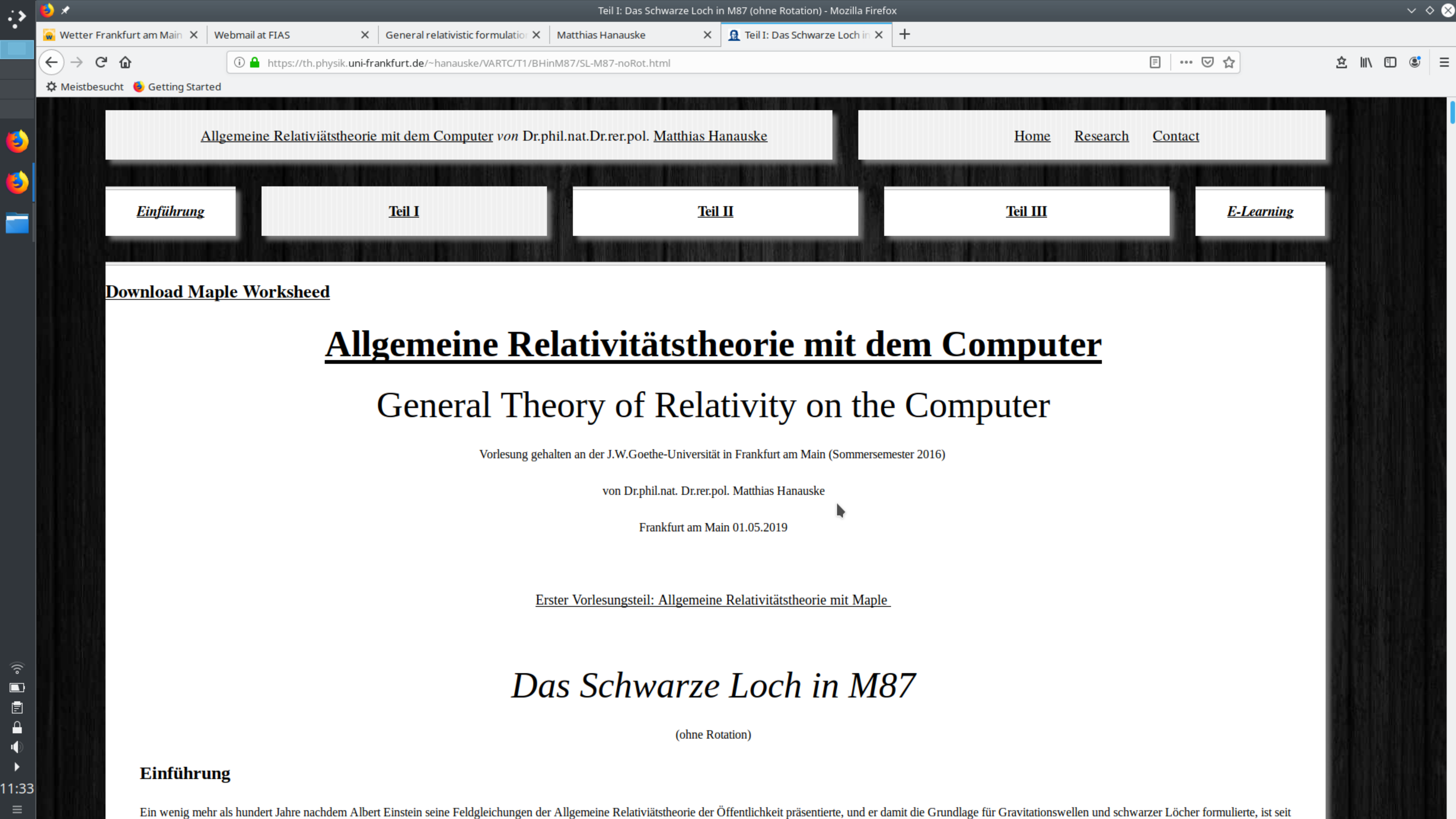
YouTube Video: https://www.youtube.com/watch?v=Zh5p9Sro_VU&list=PLn5gYfEKIag8nps1GKLqUW35AOgQY7aM2

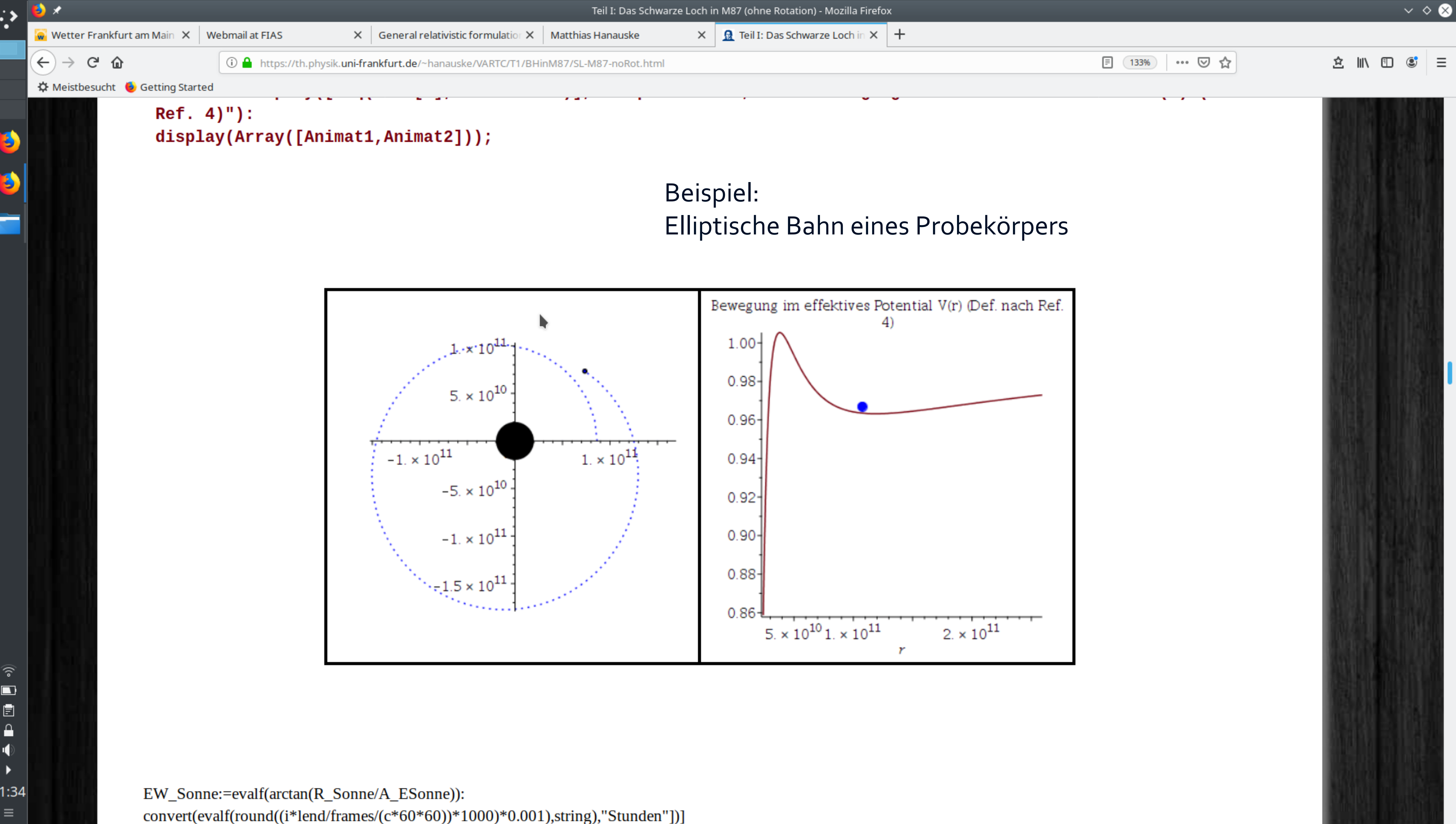
Anlässlich der bahnbrechenden Aufnahme des ersten Bildes eines schwarzen Lochs im Zentrum unserer Nachbargalaxie M87, wurde am 17. April 2019 um 20 Uhr ein öffentlicher, populärwissenschaftlicher Abendvortrag im Otto Stern Zentrum (OSZ H1) am Campus Riedberg der Goethe Universität gehalten. Es sprachen die drei Principal Investigators des europäischen Black Hole Cam-Projekts (L.Rezzolla, M.Kramer und H.Falke).

Das effektive Potential eines Probekörpers am ISCO hat eine Sattelpunktseigenschaft

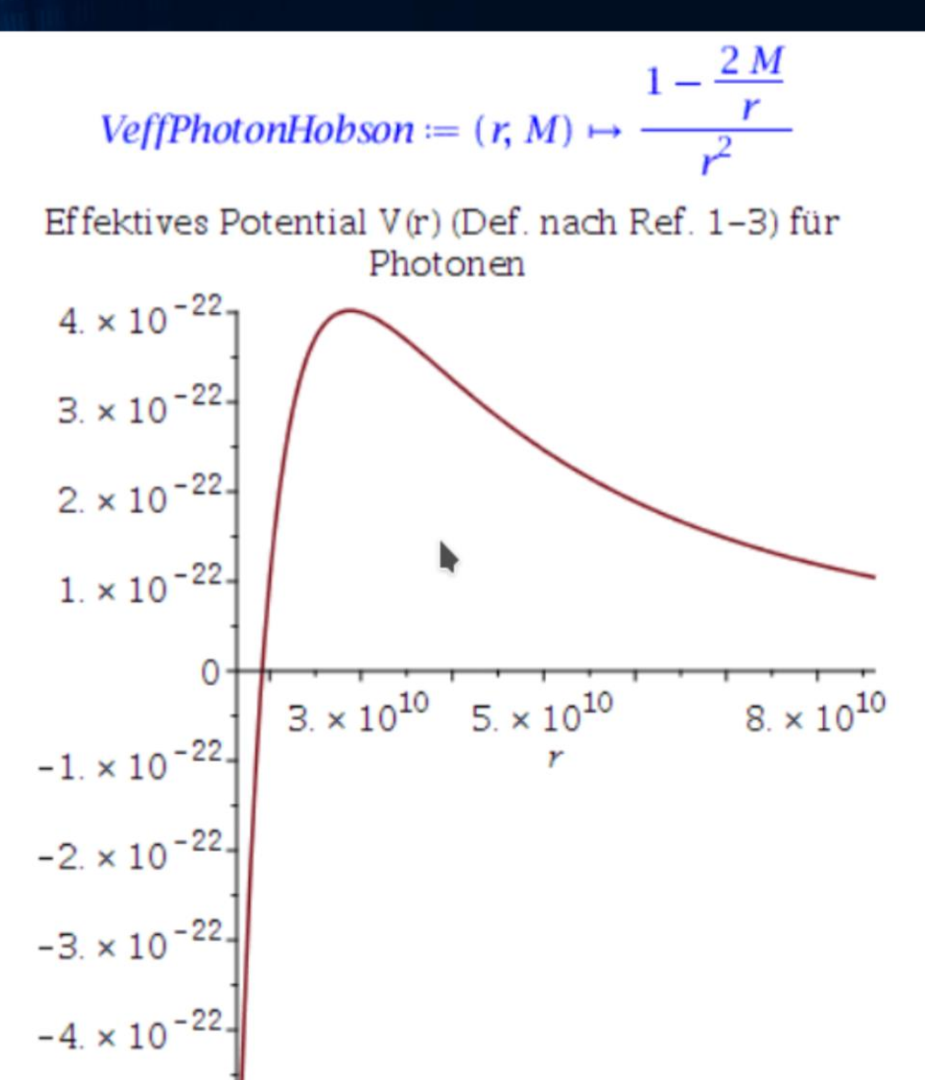





Teil I: Das Schwarze Loch in M87 (ohne Rotation) - Mozilla Firefox
Wetter Frankfurt am Main
Webmail at FIAS
General relativistic formulatio
Matthias Hanauske
Teil I: Das Schwarze Loch in
https://th.physik.uni-frankfurt.de/~hanauske/VARTC/T1/BHinM87/SL-M87-noRot.html
Meistbesucht
Getting Started
Allgemeine Relativiätstheorie mit dem Computer von Dr.phil.nat.Dr.rer.pol. Matthias Hanauske
Home
Research
Contact
Einführung
Teil I
Teil II
Teil III
E-Learning
Download Maple Worksheed
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 2016)
von Dr.phil.nat. Dr.rer.pol. Matthias Hanauske
Frankfurt am Main 01.05.2019
Erster Vorlesungsteil: Allgemeine Relativitätstheorie mit Maple
Das Schwarze Loch in M87
(ohne Rotation)
Einführung
Ein wenig mehr als hundert Jahre nachdem Albert Einstein seine Feldgleichungen der Allgemeine Relativiätstheorie der Öffentlichkeit präsentierte, und er damit die Grundlage für Gravitationswellen und schwarzer Löcher formulierte, ist seit
11:33


Teil I: Das Schwarze Loch in M87 (ohne Rotation) - Mozilla Firefox
Wetter Frankfurt am Main
Webmail at FIAS
General relativistic formulatio
Matthias Hanauske
Teil I: Das Schwarze Loch in
https://th.physik.uni-frankfurt.de/~hanauske/VARTC/T1/BHinM87/SL-M87-noRot.html
133%
Meistbesucht
Getting Started
Ref. 4)"):
display(Array([Animat1,Animat2]));
Beispiel:
Elliptische Bahn eines Probekörpers
1. × 10^11
5. × 10^10
-1. × 10^11
1. × 10^11
-5. × 10^10
-1. × 10^11
-1.5 × 10^11
Bewegung im effektives Potential V(r) (Def. nach Ref. 4)
1.00
0.98
0.96
0.94
0.92
0.90
0.88
0.86
5. × 10^10
1. × 10^11
2. × 10^11
r
EW_Sonne:=evalf(arctan(R_Sonne/A_ESonne)):
convert(evalf(round((i*lend/frames/(c*60*60))*1000)*0.001),string),"Stunden"])]
11:34

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

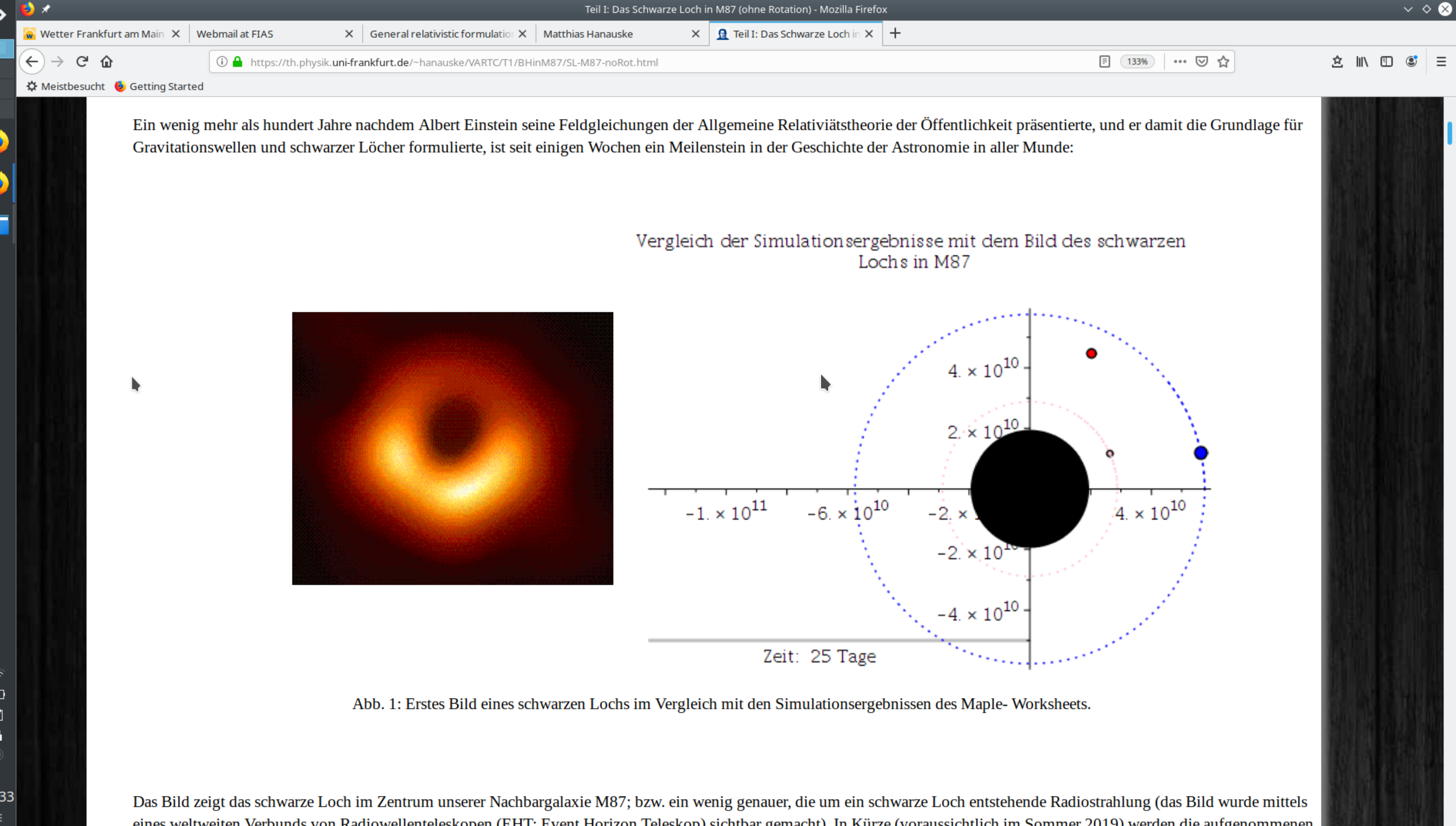


Vergleich der Simulationsergebnisse mit dem Bild des schwarzen Lochs in M87

4. × 10^10
2. × 10^10
-2. × 10^10
-4. × 10^10
-1. × 10^11
-6. × 10^10
-2. ×
4. × 10^10

Zeit: 25 Tage

Blau: ISCO , Pink: Photonensphäre



Ein wenig mehr als hundert Jahre nachdem Albert Einstein seine Feldgleichungen der Allgemeine Relativiätstheorie der Öffentlichkeit präsentierte, und er damit die Grundlage für Gravitationswellen und schwarzer Löcher formulierte, ist seit einigen Wochen ein Meilenstein in der Geschichte der Astronomie in aller Munde:

Abb. 1: Erstes Bild eines schwarzen Lochs im Vergleich mit den Simulationsergebnissen des Maple- Worksheets.

Das Bild zeigt das schwarze Loch im Zentrum unserer Nachbargalaxie M87; bzw. ein wenig genauer, die um ein schwarze Loch entstehende Radiostrahlung (das Bild wurde mittels eines weltweiten Verbunds von Radiowellenteleskopen (EHT: Event Horizon Teleskop) sichtbar gemacht). In Kürze (voraussichtlich im Sommer 2019) werden die aufgenommenen

Einführung in die Parallele Programmierung

fias.uni-frankfurt.de/~hanauske/VARTC/T2/intro/Hanauske_ParallelizationTut.odp
fias.uni-frankfurt.de/~hanauske/VARTC/T2/intro/Hanauske_ParallelizationTut.pdf

Introduction

1. Parallelization on shared memory systems using OpenMP
2. Parallelization on distributed memory systems using MPI
3. Further resources

Introduction

1. Introduction
   a) What is parallelization?
   b) When and where can it be used?
   c) Parallel architectures of computer clusters.
   d) Different parallelization languages.
2. Parallelization on shared memory systems using OpenMP
3. Parallelization on distributed memory systems using MPI
4. Futher resources

Introduction

Parallel Programming is a programming paradigm (a fundamental style of computer programming).

Within a **parallel computer code** a single computation problem is separated in different portions that may be **executed concurrently** by different processors.

Parallel Programming is a construction of a computer code that allows its execution on a **parallel computer** (multi-processor computer) in order to reduce the time needed for a single computation problem.

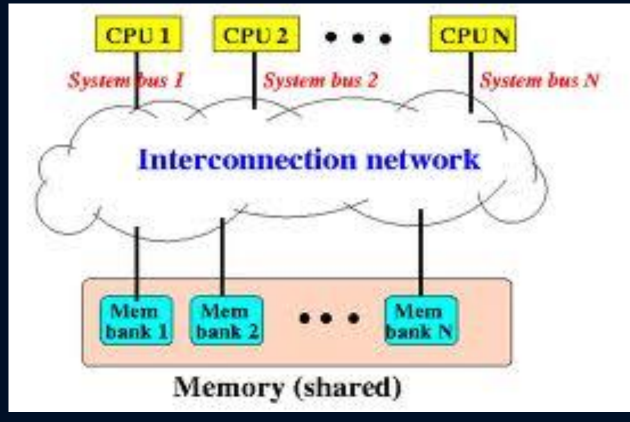
Depending on the architecture of the parallel computer (or computer cluster) the **Parallel Programming Framework** (OpenMP, MPI, Cuda, OpenCL, …) has to be choosen .

Parallel computers

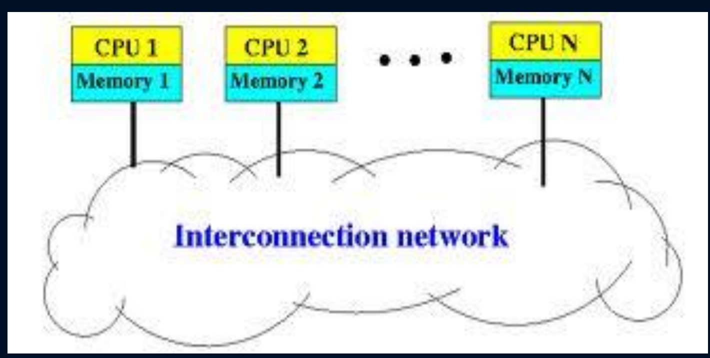
Parallel computer architectures:

SIMD (Single Instruction, Multiple Data)
Example: Parallel computers with graphics processing units (GPU's) using the Cuda or OpenCL language.

MIMD (Multiple Instruction, Multiple Data)



Shared Memory
(OpenMP, OpenCL, MPI)



Distributed Memory
(MPI, (Shell programming))

Performance

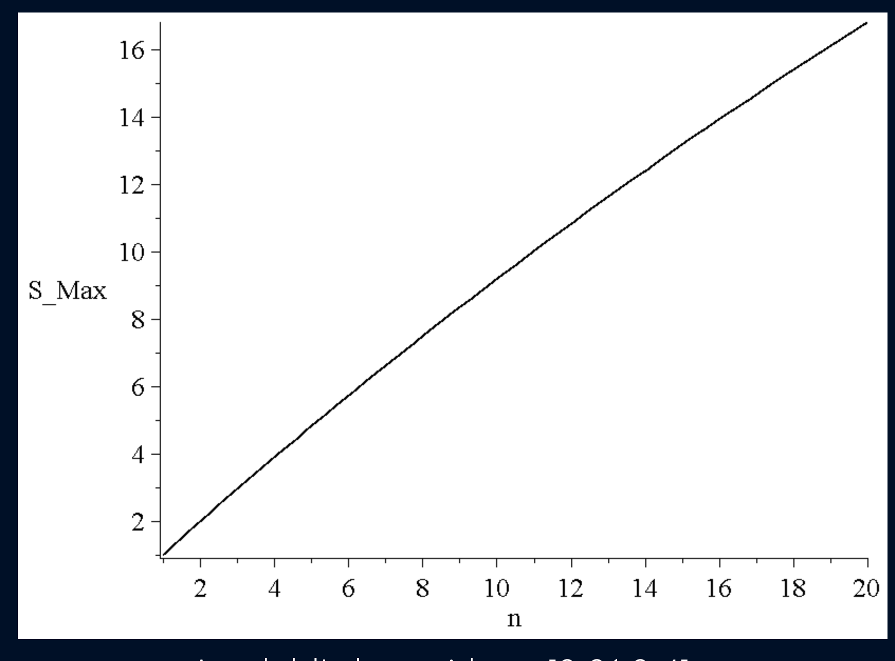
The **performance of a parallel computer code** can be measured using the following characteristic values:

T(n): Time needed to run the programm on n processing elements (e.g. CPU's, computer nodes).
Speedup: $S(n):=T(1)/T(n)$, **Efficiency**: $E(n):=S(n)/n$

Amdahl's law:

The "Amdahl's law" describes the speedup of an optimal parallel computer code. T(n) is diveded in two parts ($T(n)=Ts+Tp(n)$), where Ts is the time needed for the non-parallizable part of the programm and Tp(n) is the parallizable part, which can be executed concurrently by different processors. $A(n):=Max(S(n))$

$A(n)=1/(a+(1-a)/n)$, with $a:=Ts/T(1)$



Amdahl's law with a=[0.01,0.4]

Shared Memory and OpenMP

1. Introduction
2. Parallelization on shared memory systems using OpenMP
   a) Introduction to OpenMP
   b) Example
   c) Further OpenMP directives
   d) Additional material
3. Parallelization on distributed memory systems using MPI
4. Further resources

OpenMP

The **parallel computer language "OpenMP** (Open Multi-Processing)" supports multi-platform shared-memory parallel programming in C/C++ and Fortran. It is a collaborative developed parallel language which has its origin in 1997.

OpenMP separates the parallizable part of the program into several **'Threads'** where each thread can be executed on a different processing element (CPU) using shared memory.

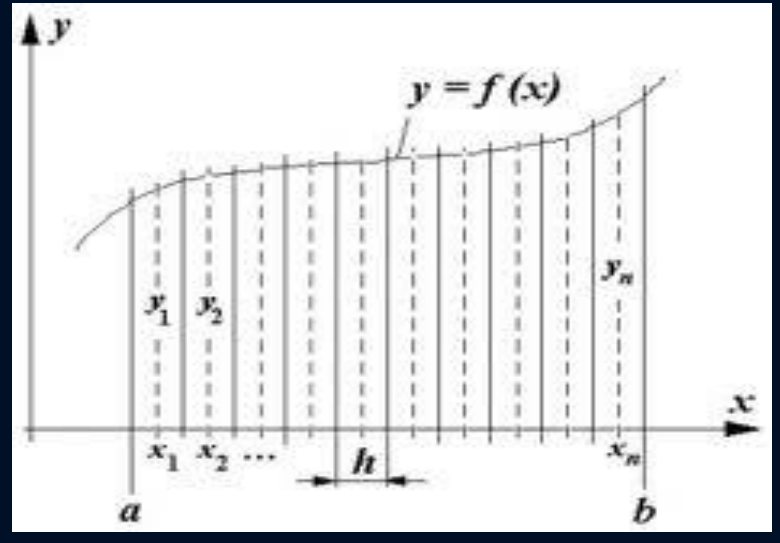
OpenMP has the advantage that common **sequential codes can easily be changed** by simply adding some OpenMP directives. Another feature of OpenMP is that the program runs also properly (but then sequentially, using only one thread) even if the compiler does not know OpenMP.

Example

The simple computation problem used in the following is the nummerical integration of an integral using the Gauss integration method. The following integral should be calculated for 10 different values (a=1,2,..,10).

$$\int_0^1 \frac{1}{1 + a x^2} \, dx = \frac{\arctan(\sqrt{a})}{\sqrt{a}}$$

The integration interval [0,1] is diveded into N pieces. The value of the integration function is taken at the middle of each integration segment (Gauss method).



Gauss'schen integration method.

C++ Einführung
Grundgerüst und Variablen

Einlesen von Header-Files (Definition nötiger C++ Funktionen)

Beginn des Hauptprogramms

Ausgabe eines Strings

```
#include <iostream>           //Ein-/Ausgabe (Include-Dateien)
using namespace std;          //Fuer cout

main(void)   //Hauptprogramm
  {

    cout<<"Hello \n";

  }
```

Prog1.cpp

Deklaration einer Integer (natürliche Zahl) und einer double (reelle Zahl) Variable

Variablen bekommen einen festen Zahlenwert (Initialisierung)

Ausgabe des Wertes der Variablen

```
#include <iostream>           //Ein-/Ausgabe (Include-Dateien)
using namespace std;          //Fuer cout

main(void)   //Hauptprogramm
  {
    //Variablendeklarationen
    int i;
    double a;

    //Variableninitialisierung
    i=3;
    a=1.435553;

    cout<<"i="<<i<<"\n";
    cout<<"a="<<a<<"\n";

  }
```

Vom Quellcode zum ausführbaren Programm

Der Quellcode (z.B. Prog1.cpp) muss compiliert werden um ein ausführbares Programm (a.out) zu erzeugen. Man öffnet hierzu in dem Verzeichnis in dem sich der Quellcode befindet, ein Terminal und führt das folgende Kommando aus:

c++ Prog1.cpp

```
hanauske@ITPRelAstro-Aspire-VN7-591G:~$ c++ Prog1.cpp
hanauske@ITPRelAstro-Aspire-VN7-591G:~$ ./a.out
Hello
hanauske@ITPRelAstro-Aspire-VN7-591G:~$
```

Beim Compilierungsprozess wird eine Datei (a.out) erzeugt, die man dann mittels des folgenden Kommandos ausführen kann:

./a.out

Das Programm wird gestartet und erzeugt im Terminal die Ausgabe "Hello"

C++ Die for-Schleife

```
#include <iostream>                    //Ein-/Ausgabe (Include-Dateien)
using namespace std;                   //Fuer cout

main(void)  //Hauptprogramm
  {
    //Variablendeklarationen
    int i;
    double a;

    //Variableninitialisierung
    a=1.435553;

    //for Schleife
    for (i = 1; i <= 10; ++i)
      {
        cout<<"i="<<i<<"\n";
        cout<<"i mal a ="<<i*a<<"\n";
      }
  }
```

Mittels einer for-Schleife können iterative Aufgaben im Programm implementiert werden. Die for-Schleife benötigt einen Anfangswert (i=0), die Angabe wie lange sie die Interation dürchführen soll (i<=10) und die Angabe um wieviel sie die Variable in jedem Schritt verändern soll (++i). “++i” bzw. “i++” ist nur eine Kurzschreibweise von i=i+1.

C++ Die do-Schleife

```
#include <iostream>                    //Ein-/Ausgabe (Include-Dateien)
using namespace std;                   //Fuer cout

main(void)  //Hauptprogramm
  {
    //Variablendeklarationen
    int i;
    double a;

    //Variableninitialisierung
    i=1;
    a=1.435553;

    //do Schleife
    do
      {
        cout<<"i="<<i<<"\n";
        cout<<"i mal a ="<<i*a<<"\n";
        i++;
      }
     while(i <= 10);
  }
```

Mittels einer do-Schleife können iterative Aufgaben im Programm implementiert werden. Die do-Schleife benötigt lediglich eine Abbruchbedingung (while(i<=10);) wobei im Inneren der Schleife die Variable i in jedem Schritt verändert werden muss (i++;). Die Variable I muss jedoch zunächst außerhalb der Schleife initialisiert werden (i=1;).

Sequential Code

```
#include <stdio.h>
#include <math.h>

double integral(int N, double a)
{
    double sum=0;
    double dx = 1/(double)(N);

    for(int j=0; j <= N; ++j)
    {
        double x = dx*( j - 0.5 );
        sum += 1/(1 + a*x*x);
    }
    return dx*sum;
}

int main(void)
{
    double integral(int,double);

    for (int i = 1; i <= 10; ++i)
    {
    printf("a=%i: Integral=%e, Difference=%e \n"
        ,i,integral(10000000,i)
        ,integral(10000000,i)-atan(sqrt(i))/sqrt(i));
    }

    return 0;
}
```

The integral is defined as a function which depends on two variables (N and a). ‘N’ is the number of integration points (integration segments) and ‘a’ is the parameter defined within the example. With the use of a ‘for-loop’, the total area of the N-rectangles are summed up. The value of the integral is then returned (dx*sum).

To calculate and output the value of the integral for different values of ‘a’ (a=1,..,10), the main function of the program contains also a ‘for-loop’. The output contains the value of ‘a’, the value of the calculated integral (N=10 million) and the difference of the calculation with the ‘analytic’ result.

Parallel Code No.1

```c
#include <stdio.h>
#include <math.h>
#include <omp.h>

double integral(int N, double a)
{
    double sum=0;
    double dx = 1/(double)(N);

    for(int j=0; j <= N; ++j)
    {
        double x = dx*( j - 0.5 );
        sum += 1/(1 + a*x*x);
    }
    return dx*sum;
}

int main(void)
{
    double integral(int,double);

#pragma omp parallel for
    for (int i = 1; i <= 10; ++i)
    {
    printf("a=%i: Integral=%e, Difference=%e \n"
        ,i,integral(10000000,i)
        ,integral(10000000,i)-atan(sqrt(i))/sqrt(i));
    }

    return 0;
}
```

To parallize the code with OpenMP, only two minor changes are necessary:

1) The OpenMP-Header file (omp.h) need to be included

2) The OpenMP-Pragma (#pragma omp parallel for) should be inserted just right before the loop that we want to be calculated concurrently.

During the execution of the program (when entering the parallized loop), several threads are created. The number of threads is not specified; it depends on the number of available processors and the size of the loop.

Parallel Code No.1a

```c
#include <stdio.h>
#include <math.h>
#include <time.h>
#include <omp.h>

double integral(int N, double a)
{
    double sum=0;
    double dx = 1/(double)(N);

    for(int j=0; j <= N; ++j)
    {
        double x = dx*( j - 0.5 );
        sum += 1/(1 + a*x*x);
    }
    return dx*sum;
}

int main(void)
{
    int startTime = time(NULL);
    double integral(int,double);

#pragma omp parallel for
    for (int i = 1; i <= 10; ++i)
    {
    int id = omp_get_thread_num();
    printf("a=%i: Integral=%e, Difference=%e, Thread No:%i \n"
        ,i,integral(10000000,i)
        ,integral(100000000,i)-atan(sqrt(i))/sqrt(i),id);
    }

    printf("Time needed: %i seconds\n"
        ,(int)(time(NULL)-startTime));
    return 0;
}
```

In respect to the ongoing calculation, this version of the parallized code does not differ at all from the previous one. Nevertheless two changes have been made:

To compare the performance of the parallel version of the code with its sequential counterpart, the time needed for the calculation is also printed out.

To understand the 'Thread-based' calculation, the id-number of each Thread is additionally printed out.

Running the code

To run the sequential version of the code under Linux, the executable file (a.out) has been created using the c++ compiler.

```
hanauske@green:~/Vortraege/Tutorial_Parallelization/OpenMP/C++$ c++ sequential_time.cpp
hanauske@green:~/Vortraege/Tutorial_Parallelization/OpenMP/C++$ ./a.out
a=1: Integral=7.853983e-01, Difference=1.000016e-08
a=2: Integral=6.755110e-01, Difference=9.999904e-09
a=3: Integral=6.045999e-01, Difference=9.999895e-09
a=4: Integral=5.535745e-01, Difference=9.999747e-09
a=5: Integral=5.144129e-01, Difference=1.000001e-08
a=6: Integral=4.830393e-01, Difference=1.000005e-08
a=7: Integral=4.571214e-01, Difference=9.999836e-09
a=8: Integral=4.352100e-01, Difference=9.999864e-09
a=9: Integral=4.163487e-01, Difference=1.000005e-08
a=10: Integral=3.998761e-01, Difference=1.000015e-08
Time needed: 22 seconds
hanauske@green:~/Vortraege/Tutorial_Parallelization/OpenMP/C++$
```

The parallel version (No.1a) has been created using c++ with the option '-fopenmp'. The program was executed on a system with two CPU's.

```
hanauske@green:~/Vortraege/Tutorial_Parallelization/OpenMP/C++$ c++ -fopenmp parallel_omp_1_time_id.cpp
hanauske@green:~/Vortraege/Tutorial_Parallelization/OpenMP/C++$ ./a.out
a=6: Integral=4.830393e-01, Difference=1.000005e-08, Thread No:1
a=1: Integral=7.853983e-01, Difference=1.000016e-08, Thread No:0
a=7: Integral=4.571214e-01, Difference=9.999836e-09, Thread No:1
a=2: Integral=6.755110e-01, Difference=9.999904e-09, Thread No:0
a=8: Integral=4.352100e-01, Difference=9.999864e-09, Thread No:1
a=3: Integral=6.045999e-01, Difference=9.999895e-09, Thread No:0
a=9: Integral=4.163487e-01, Difference=1.000005e-08, Thread No:1
a=4: Integral=5.535745e-01, Difference=9.999747e-09, Thread No:0
a=10: Integral=3.998761e-01, Difference=1.000015e-08, Thread No:1
a=5: Integral=5.144129e-01, Difference=1.000001e-08, Thread No:0
Time needed: 10 seconds
hanauske@green:~/Vortraege/Tutorial_Parallelization/OpenMP/C++$
```

Parallel Code No.2

(wrong!)

```c
#include <stdio.h>
#include <math.h>
#include <time.h>
#include <omp.h>

double integral(int N, double a)
{
    double sum=0;
    double dx = 1/(double)(N);

#pragma omp parallel for
    for(int j=0; j <= N; ++j)
    {
        double x = dx*( j - 0.5 );
        sum += 1/(1 + a*x*x);
    }
    return dx*sum;
}

int main(void)
{
    int startTime = time(NULL);
    double integral(int,double);

    for (int i = 1; i <= 10; ++i)
    {
    printf("a=%i: Integral=%e, Difference=%e \n"
        ,i,integral(10000000,i)
        ,integral(100000000,i)-atan(sqrt(i))/sqrt(i));
    }

    printf("Time needed: %i seconds\n"
        ,(int)(time(NULL)-startTime));
    return 0;
}
```

The OpenMP-Pragma (#pragma omp parallel for) has been inserted just right before the loop that is inside the function which calculates the integral.

```
hanauske@green:~/Vortraege/Tutorial_Parallelization/Ope
a=1: Integral=2.899848e-01, Difference=-4.724802e-01
a=2: Integral=2.628621e-01, Difference=-4.165062e-01
a=3: Integral=2.221546e-01, Difference=-3.745245e-01
a=4: Integral=2.037079e-01, Difference=-3.450925e-01
a=5: Integral=1.909233e-01, Difference=-3.232404e-01
a=6: Integral=1.833300e-01, Difference=-3.038624e-01
a=7: Integral=1.679277e-01, Difference=-2.861134e-01
a=8: Integral=1.638385e-01, Difference=-2.715731e-01
a=9: Integral=1.523053e-01, Difference=-2.583911e-01
a=10: Integral=1.486715e-01, Difference=-2.531299e-01
Time needed: 25 seconds
hanauske@green:~/Vortraege/Tutorial_Parallelization/Ope
```

Two problems arise when executing the program:

1) The parallel program needs even more time then the sequential version.

2) The integrals are calculated wrong (see hudge difference to the analytic result).

Parallel Code No.2

```c
#include <stdio.h>
#include <math.h>
#include <time.h>
#include <omp.h>

double integral(int N, double a)
{
    double sum=0;
    double dx = 1/(double)(N);

#pragma omp parallel for reduction(+:sum)
    for(int j=0; j <= N; ++j)
    {
        double x = dx*( j - 0.5 );
        sum += 1/(1 + a*x*x);
    }
    return dx*sum;
}

int main(void)
{
    int startTime = time(NULL);
    double integral(int,double);

    for (int i = 1; i <= 10; ++i)
    {
    printf("a=%i: Integral=%e, Difference=%e \n"
        ,i,integral(10000000,i)
        ,integral(100000000,i)-atan(sqrt(i))/sqrt(i));
    }

    printf("Time needed: %i seconds\n"
        ,(int)(time(NULL)-startTime));
    return 0;
}
```

The problem of the previous code is due to a wrong comunication and interference between the threads at the code line

sum += 1/(1+a*x*x);

During the execution, this line is actually separated in several steps:

1) The values of sum,a and x are read.

2) The value of '1/(1+a*x*x)' is calculated and added up with the value of 'sum'.

3) The result of 2) is written as the new value of 'sum' to the adress of the variable 'sum'.

When several threads are created inside the loop, it is possible that while one thread (A) is at stage 2), another thread (B) begins at stage 1). If A writes its new value at stage 3), B is at stage 2). When B finally writes its new value at stage 3), the integration increment of A is lost. This leads to wrong results and slowdown of execution.

This 'race condition' can be solved using the syncronisation directive

reduction(+:sum)

Running the code

Sequential version:

```
hanauske@green:~/Vortraege/Tutorial_Parallelization/OpenMP/C++$ c++ sequential_time.cpp
hanauske@green:~/Vortraege/Tutorial_Parallelization/OpenMP/C++$ ./a.out
a=1: Integral=7.853983e-01, Difference=1.000016e-08
a=2: Integral=6.755110e-01, Difference=9.999904e-09
a=3: Integral=6.045999e-01, Difference=9.999895e-09
a=4: Integral=5.535745e-01, Difference=9.999747e-09
a=5: Integral=5.144129e-01, Difference=1.000001e-08
a=6: Integral=4.830393e-01, Difference=1.000005e-08
a=7: Integral=4.571214e-01, Difference=9.999836e-09
a=8: Integral=4.352100e-01, Difference=9.999864e-09
a=9: Integral=4.163487e-01, Difference=1.000005e-08
a=10: Integral=3.998761e-01, Difference=1.000015e-08
Time needed: 22 seconds
hanauske@green:~/Vortraege/Tutorial_Parallelization/OpenMP/C++$
```

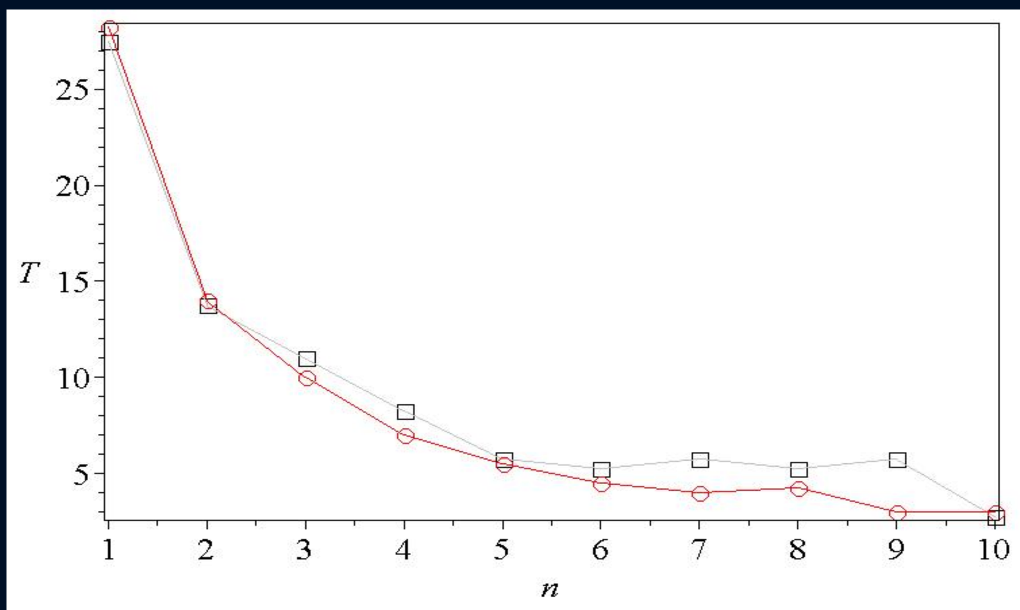
Parallel version (No.2): Due to the non-sequential summation of the different parts of the integral, a different rounding error occurs within the parallel version. This is the reason that the calculated integrals are not exactly the same as in the sequential version.

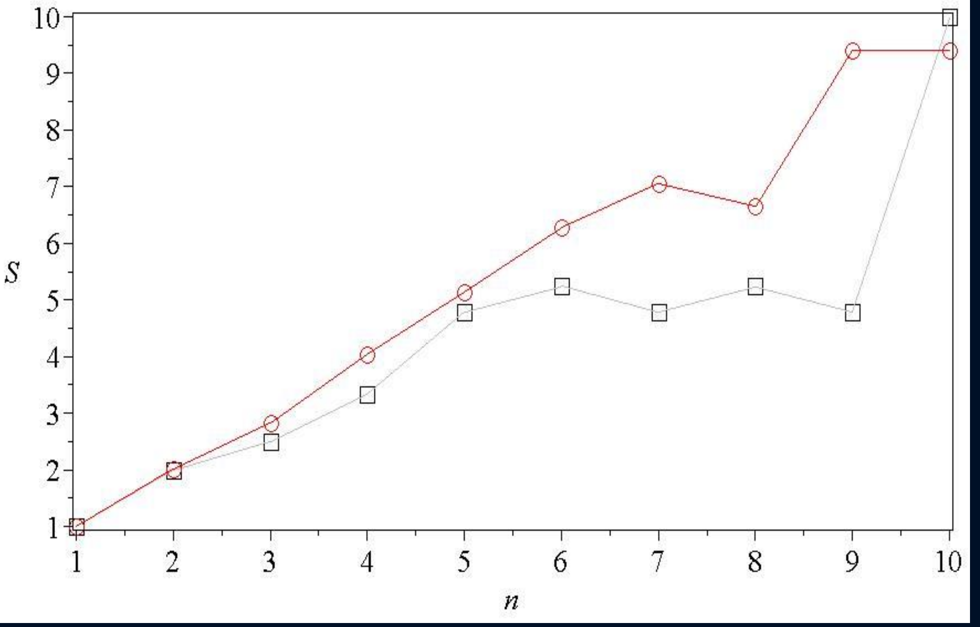
```
hanauske@green:~/Vortraege/Tutorial_Parallelization/OpenMP/C++$ c++ -fopenmp parallel_omp_2_time.cpp
hanauske@green:~/Vortraege/Tutorial_Parallelization/OpenMP/C++$ ./a.out
a=1: Integral=7.853983e-01, Difference=1.000003e-08
a=2: Integral=6.755110e-01, Difference=1.000009e-08
a=3: Integral=6.045999e-01, Difference=9.999958e-09
a=4: Integral=5.535745e-01, Difference=9.999924e-09
a=5: Integral=5.144129e-01, Difference=9.999987e-09
a=6: Integral=4.830393e-01, Difference=9.999908e-09
a=7: Integral=4.571214e-01, Difference=1.000000e-08
a=8: Integral=4.352100e-01, Difference=1.000000e-08
a=9: Integral=4.163487e-01, Difference=1.000012e-08
a=10: Integral=3.998761e-01, Difference=9.999997e-09
Time needed: 11 seconds
hanauske@green:~/Vortraege/Tutorial_Parallelization/OpenMP/C++$
```

Performance

The following calculations where performed on the Center for Scientific Computing (CSC) of the Goethe University Frankfurt using the FUCHS-CPU-Cluster.

The upper picture shows the time needed (T(n)) to run the programm using n processing elements (respectively threads) and the lower picture shows the speedup S(n). The black curve indicates the performance of the parallel code No.1 and the red curve shows the results of code No.2.





Further OpenMP directives

Loop parallelization(#pragma omp parallel for …):

- Access to variables (shared(), private(), firstprivate(), reduction())
- Syncronisation (atomic, critical)
- Locking (omp_init_lock(),omp_destroy_lock(),omp_set_lock(),_unset_lock())
- Barriers (#pragma omp barrier)
- Conditional parallelization (if(...))
- Number of threads (omp_set_num_threads())
- Loop workflow (shedule())

Additional material:

The OpenMP® API specification for parallel programming: http://openmp.org/

The Community of OpenMP: http://www.compunity.org/

OpenMP-Tutorial: https://computing.llnl.gov/tutorials/openMP/

Book: Parallel Programming in C with MPI and OpenMP, by Michael J. Quinn.

Book: Patterns for Parallel Programming, by Timothy G. Mattson, et.al.

Book: Using OpenMP, by Chapman,et.al.

Book: OpenMP by Hoffmann, Lienhart

Tutorium Examples: http://fias.uni-frankfurt.de/~hanauske/new/parallel/openmp/

Distributed Memory and MPI

1. Introduction
2. Parallelization on shared memory systems using OpenMP
3. Parallelization on distributed memory systems using MPI
   a) Introduction to MPI
   b) Example
   c) Additional material
4. Further resources

MPI

The **parallel computer language "MPI** (Message Passing Interface)" supports multi-platform shared- amd distributed-memory parallel programming in C/C++ and Fortran. The MPI standard was firstly presented at the "Supercomputing '93"-conference in 1993.

With MPI, the whole computation problem is separated in different Tasks (processes). Each process can run on a different computer nodes within a computer cluster. In contrast to OpenMP, MPI is designed to run on distributed-memory parallel computers.

As each process has its own memory, the result of the whole computation problem has to be combined by using both point-to-point and collective communication between the processes.

Parallel Code No.1

```
#include <stdio.h>
#include <math.h>
#include <time.h>
#include <mpi.h>

double integral(int N, double a)
{
    double sum=0;
    double dx = 1/(double)(N);

    for(int j=0; j <= N; ++j)
    {
        double x = dx*( j - 0.5 );
        sum += 1/(1 + a*x*x);
    }
    return dx*sum;
}

int main( int nArguments, char **arguments )
{
    MPI::Status status;
    MPI::Init ( nArguments, arguments );
    int p = MPI::COMM_WORLD.Get_size ( );
    int id = MPI::COMM_WORLD.Get_rank ( );

    double integral(int,double);
    int startTime = time(NULL);

    for (int i = id+1; i <= 10; i = i + p)
    {
        printf("a=%i: Integral=%e, Difference=%e \n"
        ,i,integral(10000000,i)
        ,integral(100000000,i)-atan(sqrt(i))/sqrt(i));
    }

    if (id == 0)
    {
        printf("Time needed: %i seconds\n"
        ,(int)(time(NULL)-startTime));
    }

    MPI::Finalize ( );
    return 0;
}
```

To parallize the code with MPI, some changes are necessary:

1) The MPI-Header file (mpi.h) need to be included.

2) Arguments has to be included within the main function.

3) Several other things need to be specified

At the beginning of the execution of the program a specified number of processes (p) are created. Each process has its own id-number and it can be executed on different nodes within a computer cluster or on different processors of one node. Within this version of the parallel program the loop which goes over different values of 'a' (a=1,..,10) is diveded among different processes.

Running the code No.1

To build the executable file (a.out) of the parallel version of the MPI-program under Linux, a mpi-compiler (mpic++, mpicxx, mpicc,…) has to be used. To run the program, on needs to use the command "mpirun" and specify the number of processes (e.g. -np 2).

The first run on the right hand side was performed by only using one process (sequential version).

The second run was much faster and has used two processes.

```
hanauske@green:~/Vortraege/Tutorial_Parallelization/MPI$ mpic++ parallel_mpi_time.cpp
hanauske@green:~/Vortraege/Tutorial_Parallelization/MPI$ mpirun -np 1 ./a.out
a=1: Integral=7.853983e-01, Difference=1.000016e-08
a=2: Integral=6.755110e-01, Difference=9.999904e-09
a=3: Integral=6.045999e-01, Difference=9.999895e-09
a=4: Integral=5.535745e-01, Difference=9.999747e-09
a=5: Integral=5.144129e-01, Difference=1.000001e-08
a=6: Integral=4.830393e-01, Difference=1.000005e-08
a=7: Integral=4.571214e-01, Difference=9.999836e-09
a=8: Integral=4.352100e-01, Difference=9.999864e-09
a=9: Integral=4.163487e-01, Difference=1.000005e-08
a=10: Integral=3.998761e-01, Difference=1.000015e-08
Time needed: 21 seconds
hanauske@green:~/Vortraege/Tutorial_Parallelization/MPI$ mpirun -np 2 ./a.out
a=2: Integral=6.755110e-01, Difference=9.999904e-09
a=1: Integral=7.853983e-01, Difference=1.000016e-08
a=4: Integral=5.535745e-01, Difference=9.999747e-09
a=3: Integral=6.045999e-01, Difference=9.999895e-09
a=6: Integral=4.830393e-01, Difference=1.000005e-08
a=5: Integral=5.144129e-01, Difference=1.000001e-08
a=8: Integral=4.352100e-01, Difference=9.999864e-09
a=7: Integral=4.571214e-01, Difference=9.999836e-09
a=10: Integral=3.998761e-01, Difference=1.000015e-08
a=9: Integral=4.163487e-01, Difference=1.000005e-08
Time needed: 10 seconds
```

Parallel Code No.2

```
#include <stdio.h>
#include <math.h>
#include <time.h>
#include <mpi.h>

double integral(int N, double a, int id, int p)
{
    double sum=0;
    double dx = 1/(double)(N);

    for(int j=id; j <= N; j = j + p)
    {
        double x = dx*( j - 0.5 );
        sum += 1/(1 + a*x*x);
    }
    return dx*sum;
}

int main( int nArguments, char **arguments )
{
    MPI::Status status;
    MPI::Init ( nArguments, arguments );
    int p = MPI::COMM_WORLD.Get_size ( );
    int id = MPI::COMM_WORLD.Get_rank ( );

    double integral(int,double,int,int);
    double ergebnis[24];
    int startTime = time(NULL);

    ergebnis[id]=integral(500000000,1,id,p);
    if(id != 0){MPI::COMM_WORLD.Send( &ergebnis[id], 1, MPI::DOUBLE, 0, 1);}

    if (id == 0)
    {
        for (int q = 1; q <= p - 1; q++ )
        {
            MPI::COMM_WORLD.Recv( &ergebnis[q], 1, MPI::DOUBLE, q, 1, status );
            ergebnis[0]=ergebnis[0]+ergebnis[q];
         }
        printf("a=1: Integral=%e, Difference=%e \n"
        ,ergebnis[0],ergebnis[0]-atan(sqrt(1))/sqrt(1));
        printf("Time needed: %i seconds\n",(int)(time(NULL)-startTime));
    }

    MPI::Finalize ( );
    return 0;
}
```

Within this parallel version only one integral was calculated (a=1), but the accuracy of the performed numerical calculation has been increased (N=500 million). The loop that performs these 500 milion iterations is diveded among different processes.

As every process nows only the part that it has calculated, the processes need to comunicate in order to calculate the value of the whole integral. Within MPI, several way of communications are possible. Within this version a point-to-point comunication has been used.

The MPI function “Send” was used by every process (except process 0) to send its value to process 0.

Process 0 then receives all the different values, makes a sum and prints the final result out.

One can use collective operation “MPI_Reduce” for this purpose.

Running the code No.2

To build the executable file (a.out) of the parallel version of the MPI-program under Linux, a mpi-compiler (mpic++, mpicxx, mpicc,…) has to be used. To run the program, on needs to use the command "mpirun" and specify the number of processes (e.g. -np 2). The first run on the right hand side was performed by only using one process (sequential version). The second run was much faster and has used two processes.

```
hanauske@green:~/Vortraege/Tutorial_Parallelization/MPI$ mpic++ parallel_mpi_2_time.cpp
hanauske@green:~/Vortraege/Tutorial_Parallelization/MPI$ mpirun -np 1 ./a.out
a=1: Integral=7.853982e-01, Difference=2.000005e-09
Time needed: 10 seconds
hanauske@green:~/Vortraege/Tutorial_Parallelization/MPI$ mpirun -np 2 ./a.out
a=1: Integral=7.853982e-01, Difference=1.999957e-09
Time needed: 5 seconds
hanauske@green:~/Vortraege/Tutorial_Parallelization/MPI$
```

Further MPI Functions

Point-to-point message-passing:

- Int MPI-Send(buff, count, MPI_type, dest, tag)

e.g. MPI::COMM_WORLD.Send(&ergebnis[id], 1, MPI::DOUBLE, 0, 1);

- Int MPI-Recv(buff, count, MPI_type, source, tag, stat)

e.g. MPI::COMM_WORLD.Recv(&ergebnis[q], 1, MPI::DOUBLE, q, 1, status);

- Collective Communication: MPI_Bcast
- Barriers: MPI_Barrier
- ....

Additional material:

MPI-Tutorial: https://computing.llnl.gov/tutorials/mpi/

Book: Parallel Programming in C with MPI and OpenMP, by Michael J. Quinn.

Book: Patterns for Parallel Programming, by Timothy G. Mattson, et.al.

MPI-Examples: http://people.sc.fsu.edu/~jburkardt/cpp_src/mpi/mpi.html

Tutorium Examples: http://fias.uni-frankfurt.de/~hanauske/new/parallel/mpi/

Die TOV Gleichungen

Allgemeine Relativiätstheorie mit dem Computer: Teil II

Grundlagen zur numerischen Lösung der Tolman-Oppenheimer-Volkoff Gleichung (einfaches Euler Verfahren)

Das Differentialgleichungssystem der Tolman-Oppenheimer-Volkoff (TOV) Gleichung besitzt das folgende Aussehen

$$\frac{dp}{dr} = -\frac{(p+e)\,(m+4\pi r^3 p)}{r\,(r-2m)} \tag{1}$$

$$\frac{dm}{dr} = 4\pi r^2 e \tag{2}$$

$$\frac{d\phi}{dr} = \frac{m+4\pi r^3 p}{r\,(r-2m)} \quad , \tag{3}$$

wobei $p = p(r)$ und $e = e(r)$ der Druck und die Energiedichte der Materie darstellen, $m = m(r)$ die radiusabhängige gravitative Masse ist und die Funktion $\phi = \phi(r)$ die 00- bzw. tt-Komponente der Metrik bestimmt ($g_{00} = e^{2\phi}$; hier bezeichnet e die Eulersche Zahl!).

TOV-Gleichungen: Numerisches Vorgehen

Eine numerische Lösung der Sterneigenschaften benötigt lediglich Gleichung (1) und (2) und geht im einfachsten Fall (einfaches Euler Verfahren) nach folgendem Schema vor:

- Man definiert die Zustandsgleichung (EOS) der Sternmaterie als eine Funktion e(p).
- Man startet im Sternzentrum $r = r_0$ und legt den Wert des zentralen Druckes $p = p_0 := p(r_0)$, der zentralen Energiedichte $e = e_0 := e(r_0)$ und der Masse $m = m_0 := m(r_0) = 0$ fest. Da die TOV Gleichung (1) bei $r_0 = 0$ singulär wird, wählt man hier einen sehr, sehr kleinen Wert für r_0 (z.B. $r_0 = 10^{-14}$).

$$r = 10^{-14}\,, \quad p = p_0\,, \quad e = e_0\,, \quad m = 0 \tag{4}$$

- Die TOV Gleichungen werden als Differenzengleichungen umgeschrieben und eine kleine Schrittweite $dr = \Delta r << 1$ wird festgelegt. In einer Schleife wird dann in jedem Radiusschritt die Druck- und Massenänderung berechnet und die jeweiligen Größen beim nächsten Schritt um diesen Faktor erhöht bzw. verringert:

$$dp \;=\; -\frac{(p+e)\,(m+4\pi r^3 p)}{r\,(r-2m)}\,dr$$

- Die TOV Gleichungen werden als Differenzengleichungen umgeschrieben und eine kleine Schrittweite $dr = \Delta r << 1$ wird festgelegt. In einer Schleife wird dann in jedem Radiusschritt die Druck- und Massenänderung berechnet und die jeweiligen Größen beim nächsten Schritt um diesen Faktor erhöht bzw. verringert:

$$\begin{aligned} dp &= -\frac{(p+e)\,(m+4\pi r^3 p)}{r\,(r-2m)}\,dr \\ dm &= 4\pi r^2 e\,dr \\ p &= p + dp \\ m &= m + dm \\ r &= r + dr \end{aligned}$$

- Im Laufe der iterativen Lösung verringert sich der Druck ständig. Die Schleife wird solange ausgeführt bis der Wert des Druckes gleich Null bzw. negativ wird (Abruchbedingung: $p \leq 0$), da an der Sternoberfläche der Druck verschwindet.

TOV-Gleichungen: Numerisches Vorgehen

C++ Lösen der TOV-Gleichung

```
#include <iostream>                       //Ein-/Ausgabe (Include-Dateien)
#include <math.h>                         //Mathematisches
using namespace std;                      //Fuer cout

//Definition der Zustandsgleichung
double eos(double p)
  {
    double e;
    e=pow(p/10,3.0/5);
    return e;
  }

main(void)  //Hauptprogramm
  {
    //Variablendeklarationen
    double M,p,e,r,dM,dp,de,dr;
    double eos(double);

    //Variableninitialisierung
    M=0;
    r=pow(10,-14);
    p=10*pow(0.0005,5.0/3);
    dr=0.000001;

    //do-while Schleife (Numerische Lösung der TOV-Gleichung)
    do
      {
        e=eos(p);                                 //Wert der Energiedichte bei momentanen Druck
        dM=4*M_PI*e*r*r*dr;                       //Massenzunahme bei momentanem r und Schrittweite dr
        dp=-(p+e)*(M+4*M_PI*r*r*r*p)/(r*(r-2*M))*dr;  //Druckzunahme bei momentanem r und Schrittweite dr (TOV-Gleichung)
        r=r+dr;                                   //momentaner Radius des Neutronensterns
        M=M+dM;                                   //momentane Masse des Neutronensterns innerhalb des Radius r
        p=p+dp;                                   //momentaner Druck des Neutronensterns innerhalb des Radius r
      }
    while(p>0);

    //Ausgabe der Masse und des Radius auf dem Bildschirm
    cout<<"Neutronensternradius [km]            = "<<r<<"\n";
    cout<<"Neutronensternmasse [Sonnenmassen] = "<<M/1.4766<<"\n";

    return 0;                                     //main beenden (Programmende)
  }
```

Die polytrope **Zustandsgleichung** ist als eine Funktion außerhalb des Hauptprogramms definiert

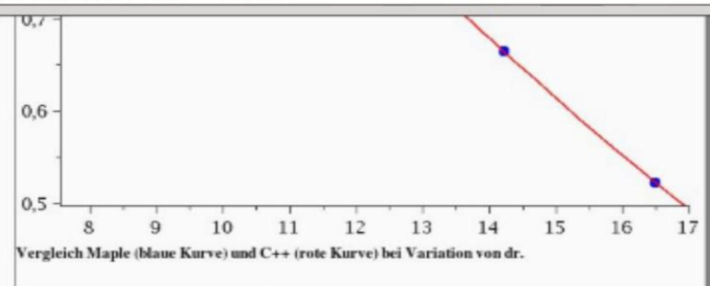
Deklaration der nötigen **Variablen** und der Zustandsgleichungsfunktion

Festlegung der **Anfangswerte** im Sternzentrum (M,r,p) und der Radiusschrittweite dr

TOV-Gleichungen

Ausgabe auf dem Bildschirm

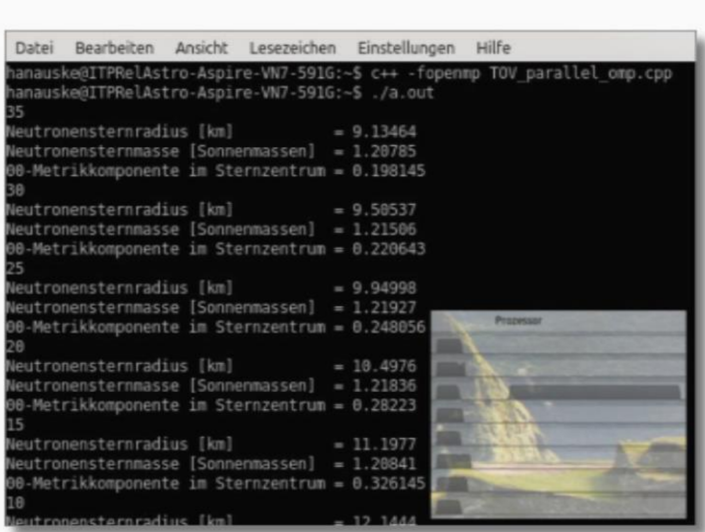
Parallele Programme siehe Teil 2 der Internetseite der Vorlesung



Vergleich Maple (blaue Kurve) und C++ (rote Kurve) bei Variation von dr.

2.3) Parallele OpenMP-Version 1 von 2.2)

Das sequentielle Programm 2.2) wurde nun mittels OpenMP (siehe TOV OpenMP Version) parallelisiert. Hiebei wurde einfach das OpenMP-Pragma '*#pragma omp parallel for private(i,M,p,e,r,nu,dM,dp,de,dnu)*' vor die for-Schleife der unabhängigen Berechnung der einzelnen Neutronensterne geschrieben. Wichtig ist nun, dass man das Programm mit dem folgenden Befehl compiliert: 'c++ -fopenmp TOV_parallel_omp.cpp'. Führt man das Programm mit './a.out' aus, so erkennt man als erstes, dass es (in Abhängigkeit wieviele CPU-Kerne man in seinem Computer hat), viel schneller läuft. Die im Terminal ausgegebenen Werte sind jedoch nicht mehr geordnet, sondern die Berechnung der 40 Neutronensterne erfolgt parallel und ungeordnet. Die nebenstehende Abbildung zeigt die Terminalausgabe des parallelen Programms und die Auslastung der 8 CPU-Kerne meines Laptops, wobei zuerst das parallele Programm und dannach das sequentielle ausgeführt wurde.

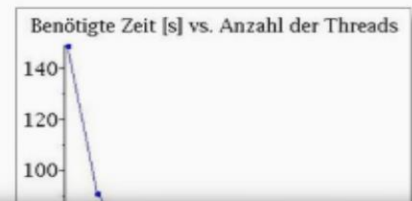


2.4) Parallele OpenMP-Version 2 (2.3) mit geordneter Terminal-Ausgabe)

Diese Version entspricht Version 2.3) mit einer geordneten Ausgabe. Die geordnete Ausgabe wird hierbei realisiert, indem für die drei ausgegebenen, numerischen Werte (Radius, Masse, zentraler g_{00}-Wert), drei Datenfelder (Arrays) der Länge 40 eingerichtet werden. Nachdem ein OpenMP-Thread mit seiner Berechnung fertig ist, speichert er sein individuelles Ergebnis in die spezifische Position innerhalb des Arrays und berechnet den nächsten Stern. Die geordnete Ausgabe aller Werte erfolgt dann sequentiell, außerhalb der parallelisierten Schleife.

2.5) Parallele OpenMP-Version 3 (2.4) mit Ausgabe in eine Datei)

Diese Version entspricht Version 2.4) wobei die geordnete Ausgabe nun nicht mehr in dem Terminal geschieht, sondern die berechneten Werte werden in eine externe Datei (tov.txt) ausgegeben - die Ausgabedatei erfolgt im Unterordner 'output', welcher vor dem Ausführen des Programms angelegt werden muss. Der Vorteil hierbei ist, dass nachdem das Programm ausgefürt wurde, die Ergebnisse einfacher verarbeitet und dargestellt werden können. So kann man z.B. mittels Gnuplot sich das Radius-Masse Diagramm darstellen. Noch einfacher, kann man sich die einzelnen Gnuplot-Befehle zum Darstellen diverser Diagramme in ein ausführbares Shell-Script schreiben, das dann automatisch die jeweiligen Plots erzeugt (siehe Gnuplot Shell-Script).



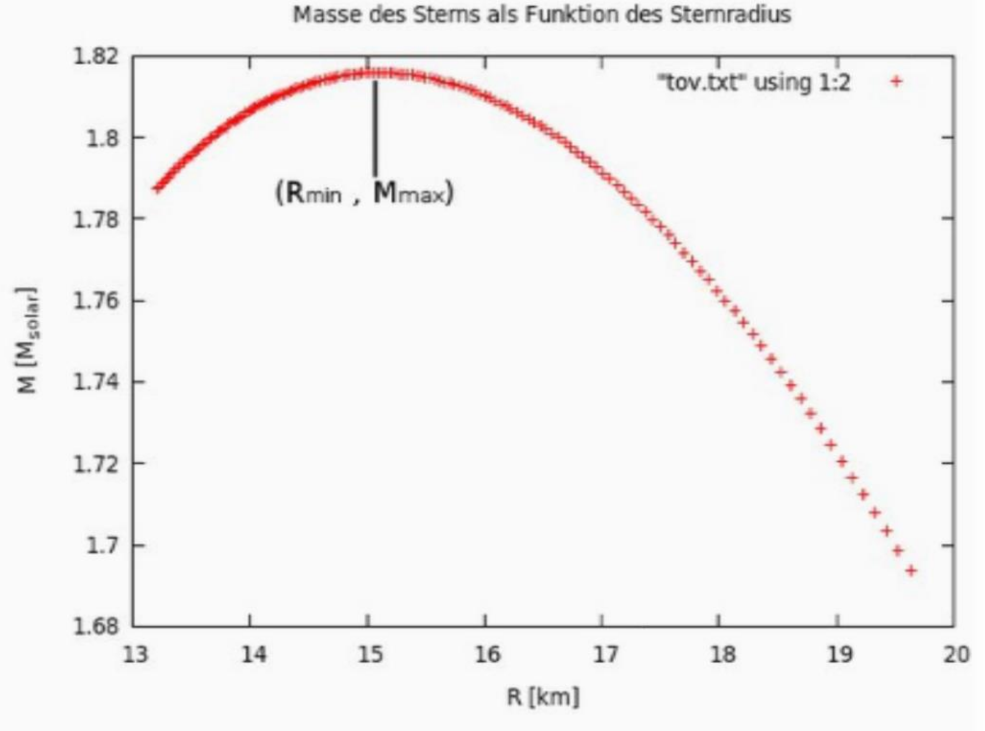
2.6) Parallele OpenMP-Version mit geordneter Ausgabe in eine Datei und variabler Zustandsgleichung

Diese Version entspricht Version 2.5), wobei die als Funktion definierte Zustandsgleichung variabler gestaltet wurde (EOS: ($e(P, K, \gamma) = (P/K)^{1/\gamma}$) für Neutronensterne und Weiße Zwerge bzw. ($e(P,Bag)=3\,p + 4\,Bag$) für Quarksterne im MIT-Bag Model).

Struktur und Performance des parallelen OpenMP - C++ Programms

Aufgaben zum 2.Teil der Vorlesung siehe E-Learning „Lon Capa"

Berechnen Sie unter Verwendung des C++ Programms aus Teil II der Vorlesung die maximale Masse M_{max} in $[\mathrm{M}_\odot]$ und den zugehörigen minimalen Radius R_{min} eines Neutronensterns in [km]. Verwenden Sie eine polytrope Zustandsgleichung der Form $p = K * e^\gamma$, wobei $\gamma = 5/3$ und $K = 20.25 \left[\mathrm{km}^{4/3}\right]$ ist.

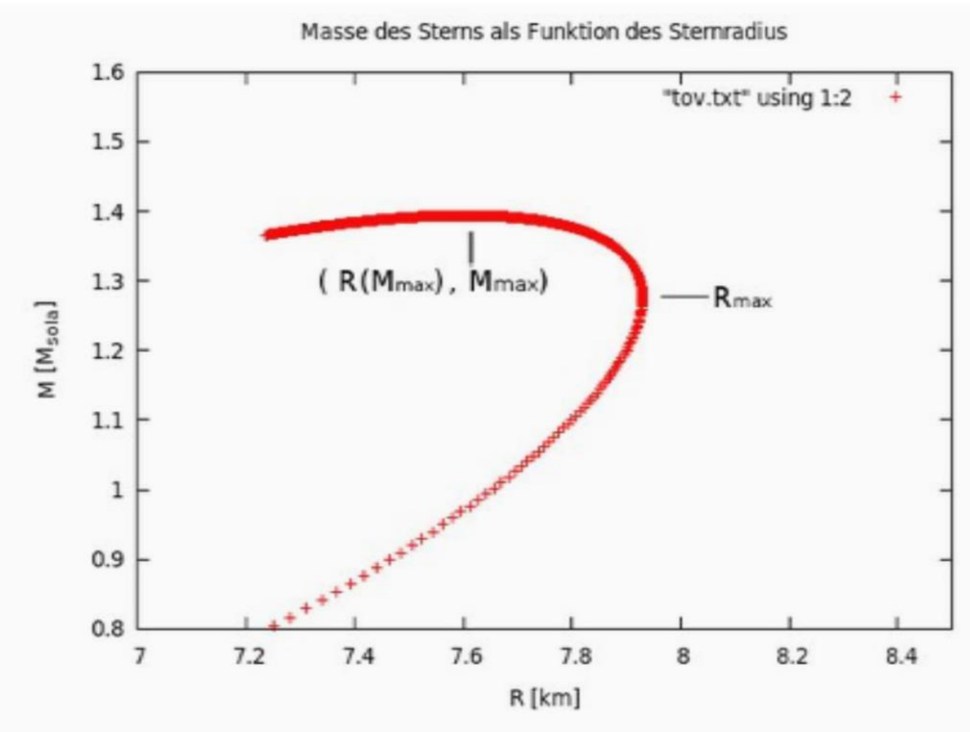


M_{max} = ______ , R_{min} = ______

Antwort einreichen   Versuche 0/20

Aufgaben zum 2.Teil der Vorlesung siehe E-Learning „Lon Capa"

Berechnen Sie unter Verwendung des C++ Programms aus Teil II der Vorlesung die maximale Masse M_{max} in $[\mathrm{M}_{\odot}]$ und den zugehörigen Radius $R(M_{max})$ eines Quarkstern Modells in [km]. Verwenden Sie die lineare Zustandsgleichung des MIT-Bag Modells $p = \frac{1}{3}\left(e - 4 * B\right)$;, wobei der Parameter B die für das Confinement nötige Bag Konstante ist; verwenden Sie B=0.000152869496944 (entspricht ungefähr $B^{1/4} =$ 194 [MeV]). Geben Sie desweiteren auch dem maximalen Radius R_{max} des Quarksternmodells an.

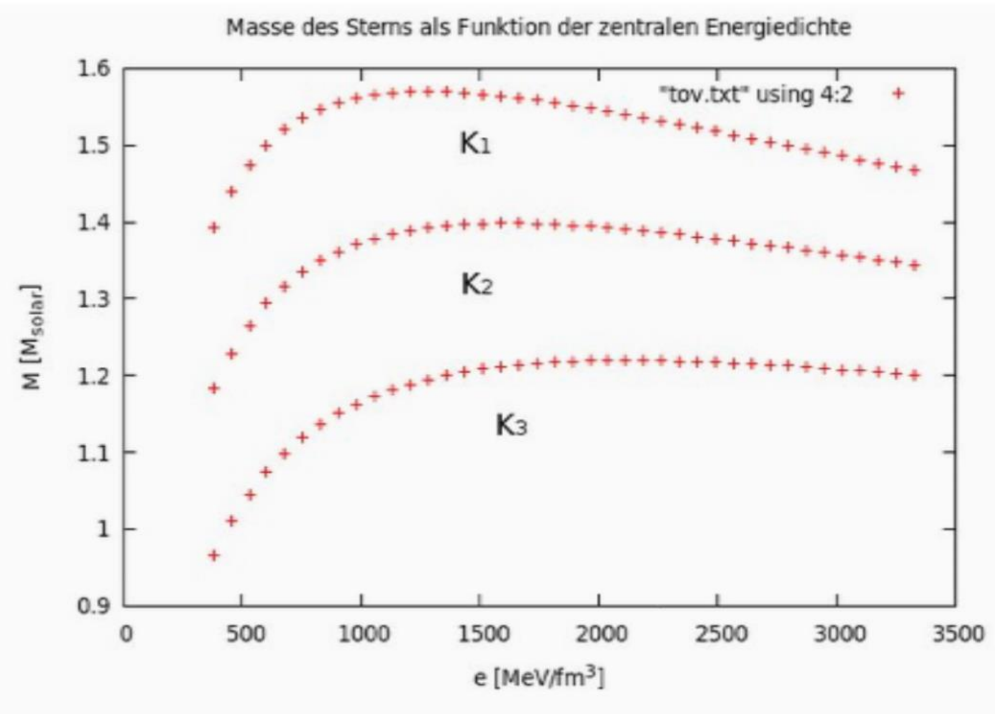


M_{max} = [ ] , $R(M_{max})$ = [ ] , R_{max} = [ ]

Antwort einreichen   Versuche 0/20

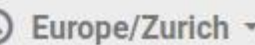
Aufgaben zum 2.Teil der Vorlesung siehe E-Learning „Lon Capa“

Die maximale Masse M_{max} eines Neutronensterns sei gegeben, und die zugrunde liegende Zustandsgleichung der Neutronensternmaterie sei durch folgenden polytropen Ansatz $p = K * e^{\gamma}$ bestimmt, wobei $\gamma = 5/3$ und $K =$ eine noch zu bestimmende unbekannte Konstante ist. Bei Variation von K ändert sich das gesamte Masse-Radius, bzw. Masse-zentrale Energiedichte Profil einer Sequenz von Sternen und der Wert der maximale Masse M_{max} verschiebt sich (siehe nebenstehende Abbildung). Berechnen Sie unter Verwendung des C++ Programms aus Teil II der Vorlesung den Wert der Konstanten K in $\left[\mathrm{km}^{4/3}\right]$ und geben Sie den zugehörigen Radius des maximalen Massen Sterns ($R_{M_{max}}$) an. Der Wert der maximalen Masse beträgt $M_{max} = 1.735147 \left[\mathrm{M}_{\odot}\right]$.

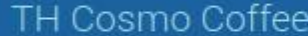


$K =$ ______ , $R_{M_{max}} =$ ______

Antwort einreichen Versuche 0/20



Europe/Zurich



Postmerger Gravitational-Wave Signatures of Phase Transitions in Binary Compact Star Mergers

by Matthias Hanauske

→ 18:30 Europe/Zurich

Description With the first detection of gravitational waves from a binary system of neutron stars GW170817, a new window was opened to study the properties of matter at and above nuclear-saturation density. Reaching densities a few times that of nuclear matter and temperatures up to 100 MeV, such mergers also represent potential sites for a phase transition (PT) from confined hadronic matter to deconfined quark matter. Especially during the postmerger evolution of the produced hypermassive/supramassive hybrid star (HMHS/SMHS) the occurrence of a "delayed PT" might give a clear gravitational wave signature of the production of quark matter in the present Universe, if the PT is strong enough. The appearance of a hadron to quark PT in the interior region of the HMHS/SMHS and its conjunction with the spectral properties of the emitted gravitational wave will be in the focus of this talk. The presented results are based on fully general-relativistic hydrodynamic simulations and employing several suitably constructed equation of states that include a PT (see Phys.Rev.Lett. 122, 061101 (2019), Phys.Rev.Lett. 124, 171103 (2020)).

Zoom connection:
https://cern.zoom.us/j/99377496227
Meeting ID: 993 7749 6227
Password: 636648