Allgemeine Relativitätstheorie mit dem Computer

PC-POOL RAUM 01.120
JOHANN WOLFGANG GOETHE UNIVERSITÄT
9. JUNI, 2017

MATTHIAS HANAUSKE

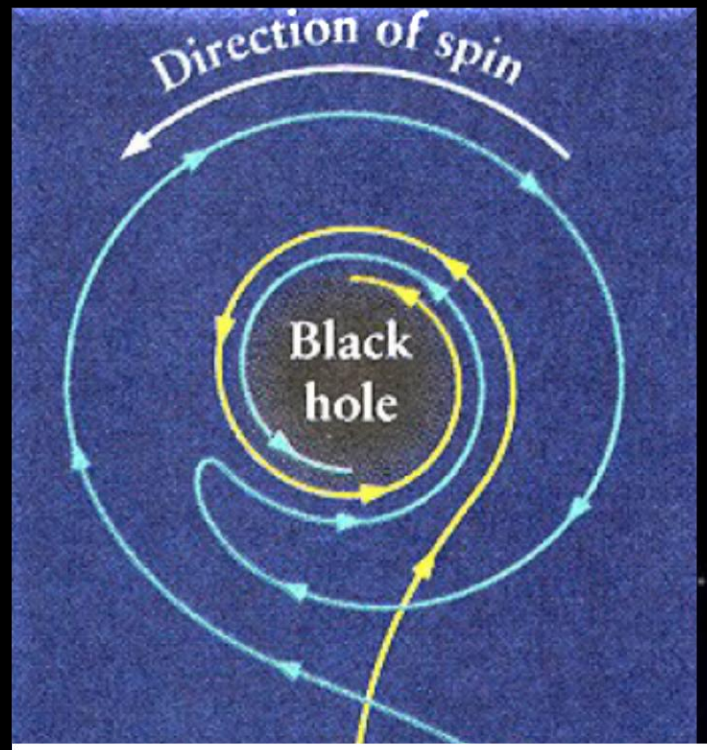
FRANKFURT INSTITUTE FOR ADVANCED STUDIES
JOHANN WOLFGANG GOETHE UNIVERSITÄT
INSTITUT FÜR THEORETISCHE PHYSIK
ARBEITSGRUPPE RELATIVISTISCHE ASTROPHYSIK
D-60438 FRANKFURT AM MAIN
GERMANY

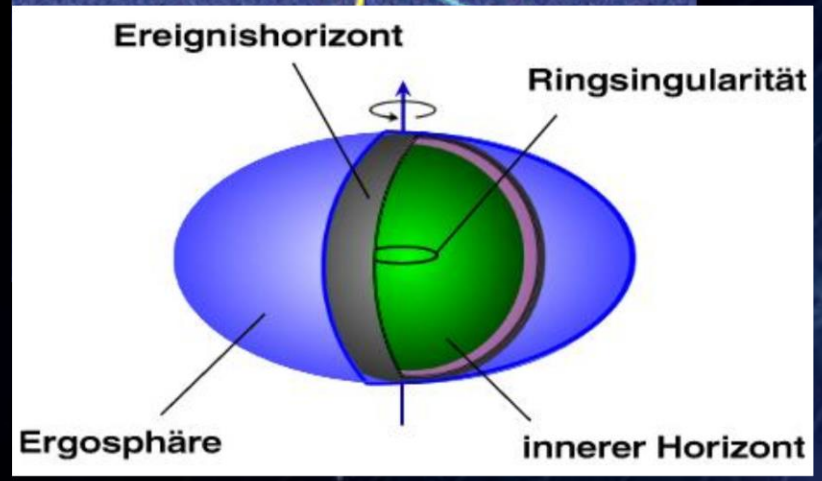
7. Vorlesung

Allgemeines zur Vorlesung, Plan für die heutige Vorlesung

- Kompensationstermine der Vorlesungen 26.05.2017 und 14.07.2017: ebenfalls verschoben werden. Mögliche neue Termine (nachmittags am Di. 20.06.2017, Di. 27.06.2017 und Di. 04.07.2017) .
- Wiederholung der letzten Vorlesung (Kerr-Metrik eines rotierenden schwarzen Loches: Ereignishorizonte und Flächen der stationären Grenze (bzw. der unendlichen Rotverschiebung), der Mitführungseffekt der Raumzeit ("Frame-Dragging"), geodätische Bewegung eines Probekörpers in der Kerr Metrik, Klassifizierung der möglichen Bahnbewegungen um ein rotierendes schwarzes Loch mittels eines effektiven Potentials)
- Weitere Aspekte der Kerr-Matrik: 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, der gravitomagnetische Effekt
- Projektarbeit: Kerr-Metrik (Geodätische Bewegung von Licht, nicht-äquatoriale Bewegung eines Probekörpers in der Kerr-Metrik, …)

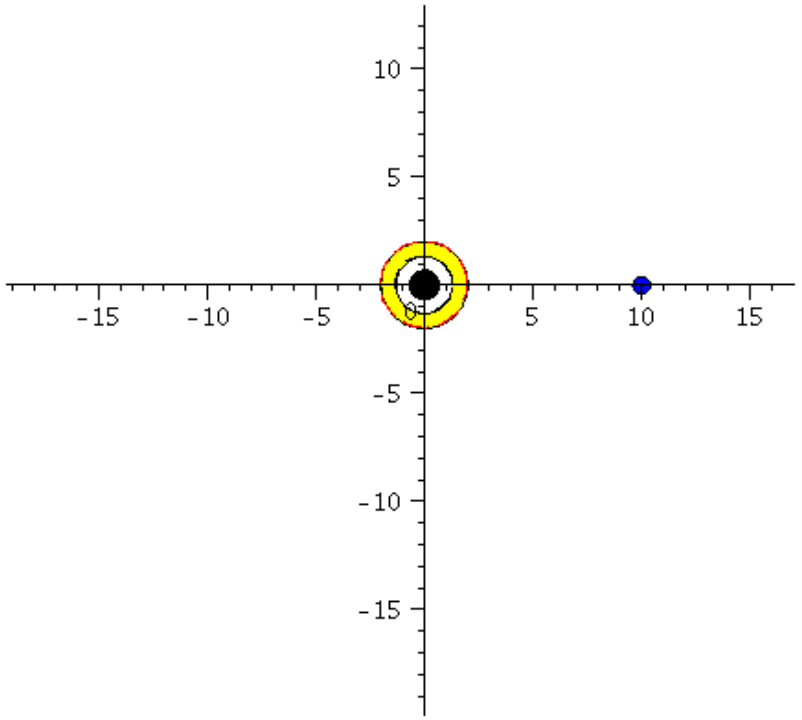
Rotierende schwarze Löcher und die Kerr Metrik

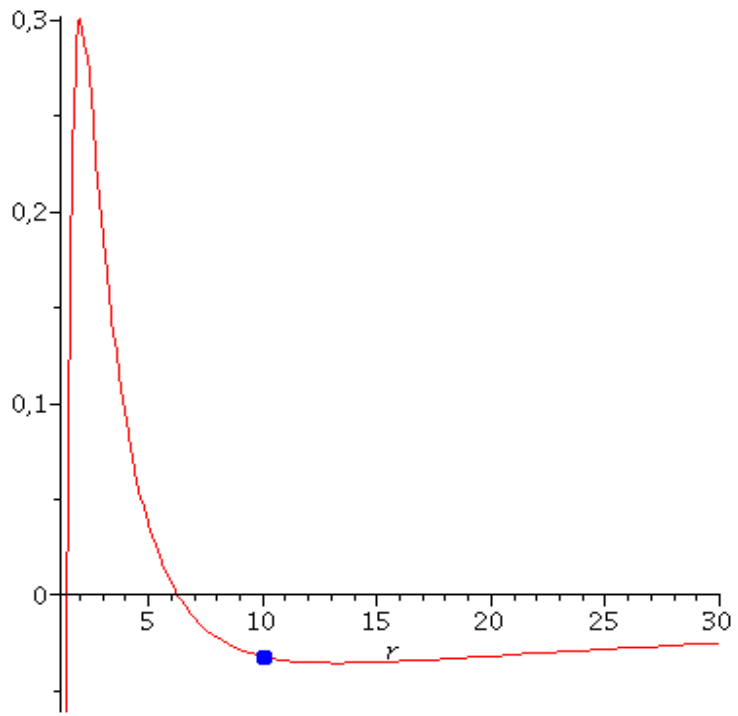




Kerr Metrik: Effektives Potential

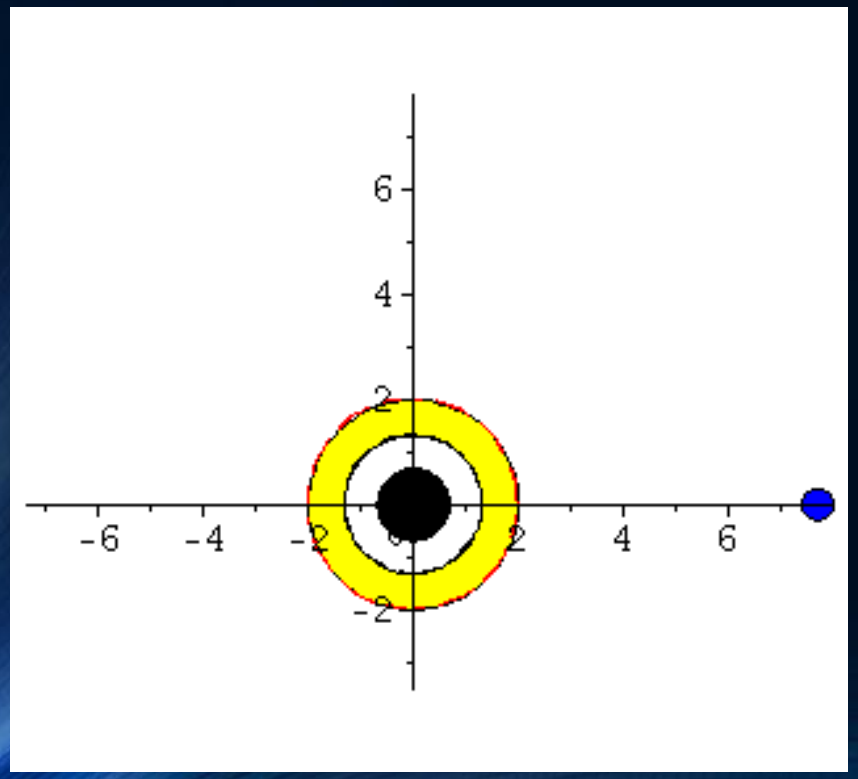
$$V_{eff}(r, M, l, a, E) = -\frac{M}{r} + \frac{l^2 - a^2\left(E^2 - 1\right)}{2\,r^2} - \frac{M(l - a\,E)^2}{r^3}$$

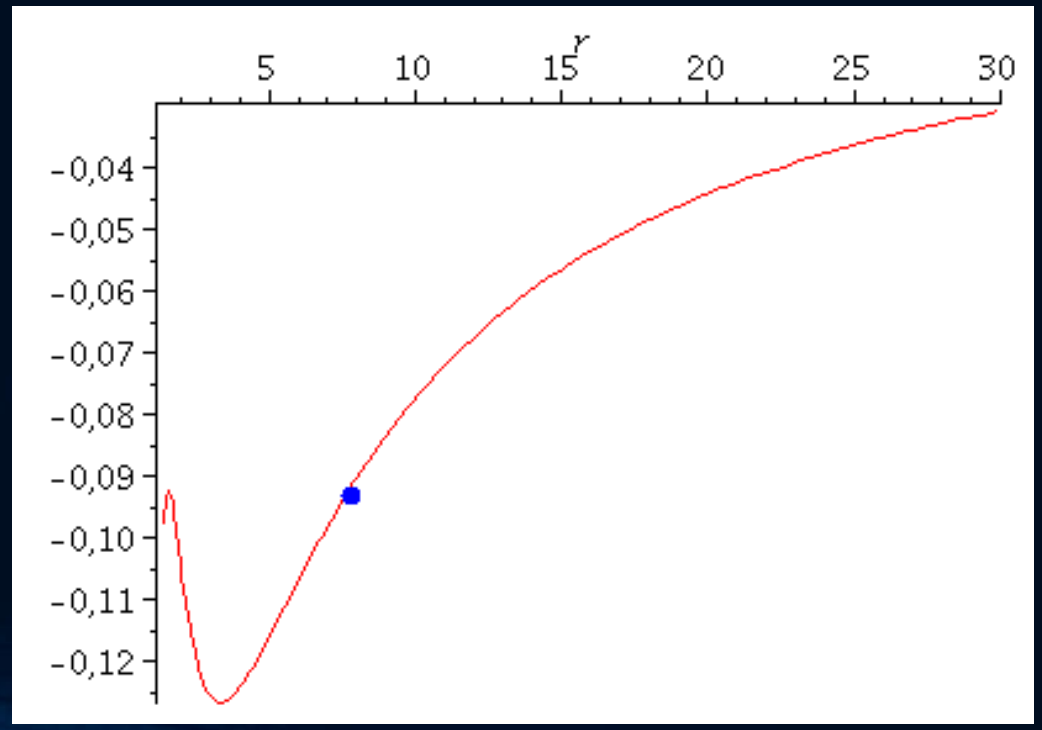




Kerr Metrik: Effektives Potential

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

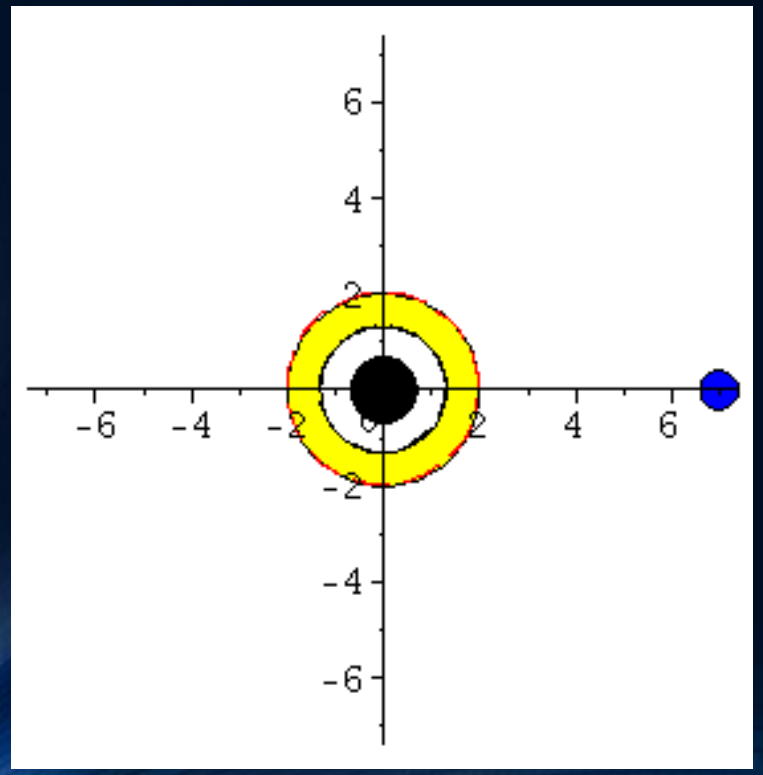


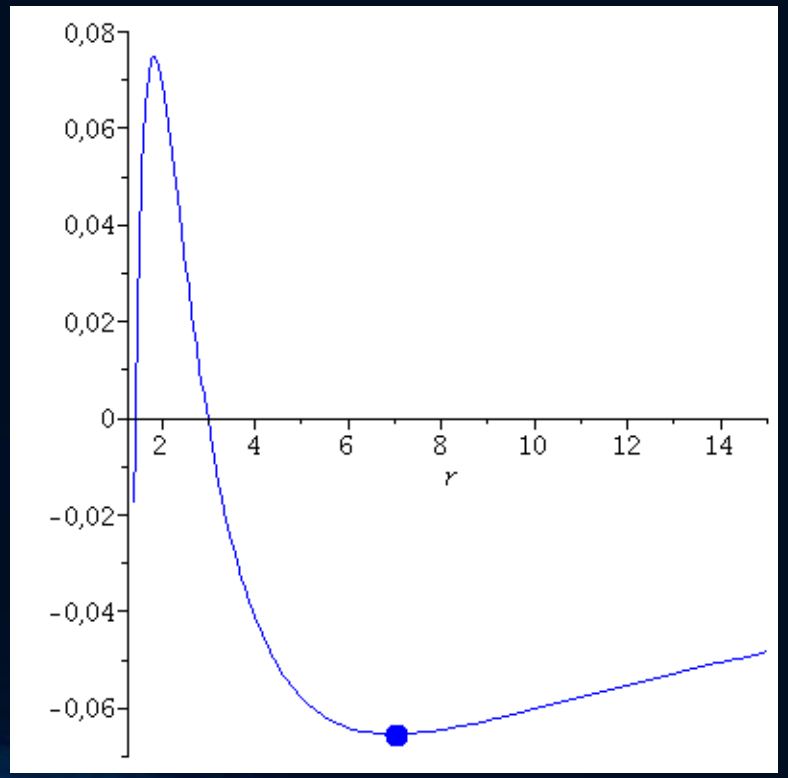


Kerr Metrik: Effektives Potential

Kreisförmige Bahnbewegungen

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

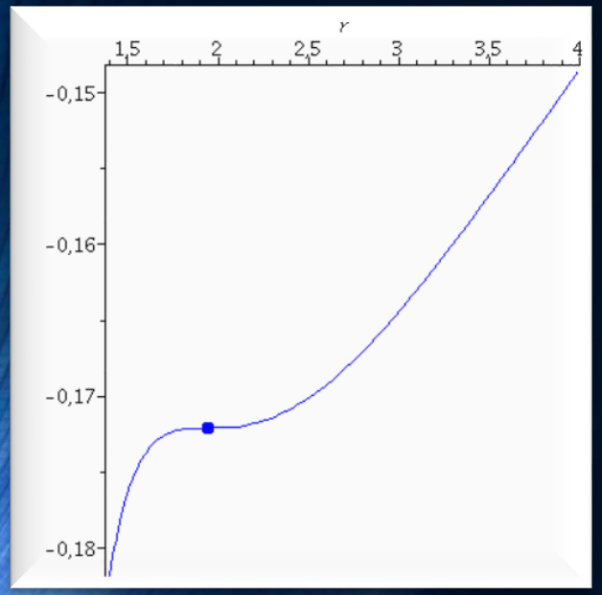




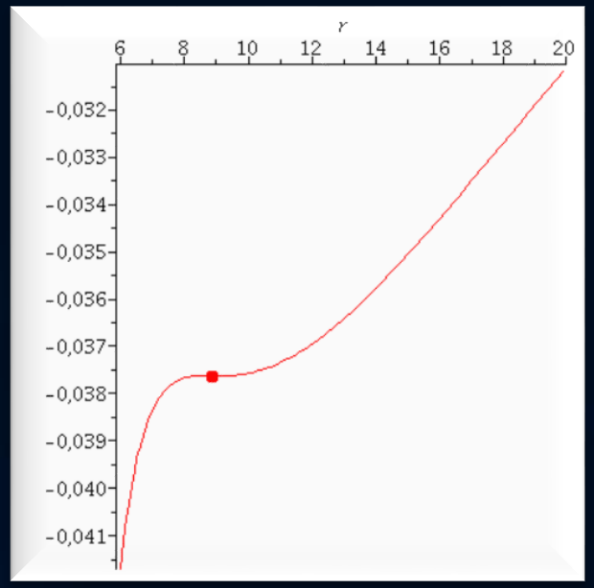
Kerr Metrik: Effektives Potential

Innerste „stabile“ kreisförmige Bahnbewegungen (ISCOs)

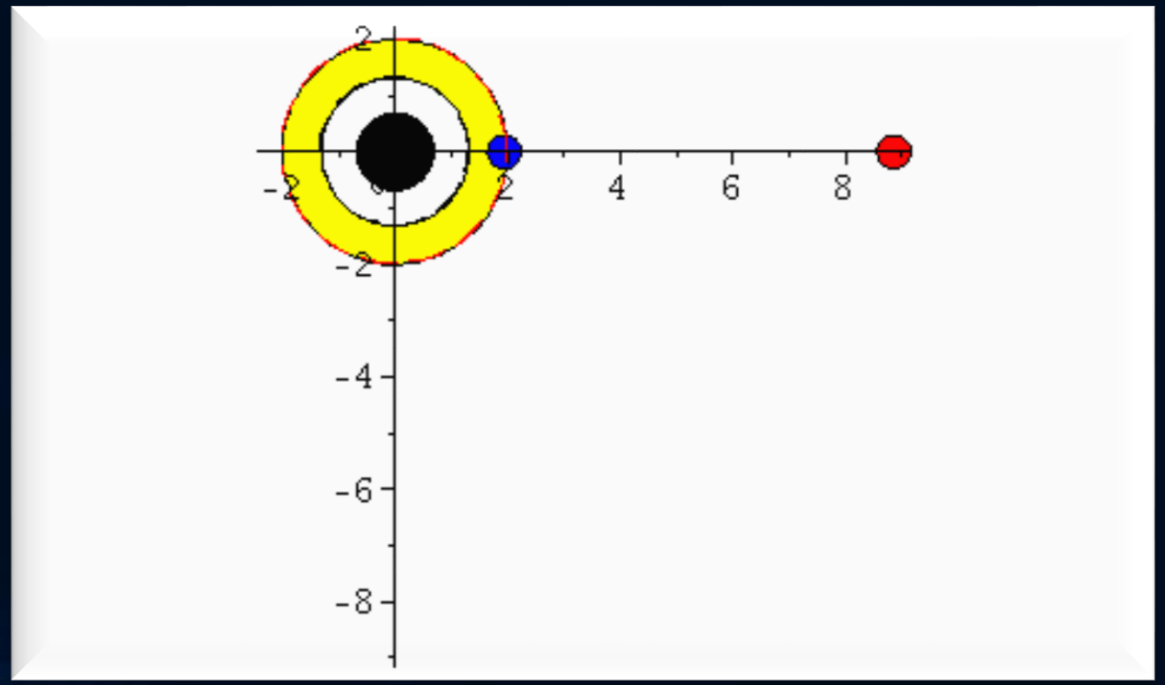
$$V_{eff}(r, M, l, a, E) = -\frac{M}{r} + \frac{l^2 - a^2\left(E^2 - 1\right)}{2\,r^2} - \frac{M(l - a\,E)^2}{r^3}$$



Probekörper rotiert mit der Rotationsrichtung des schwarzen Loches

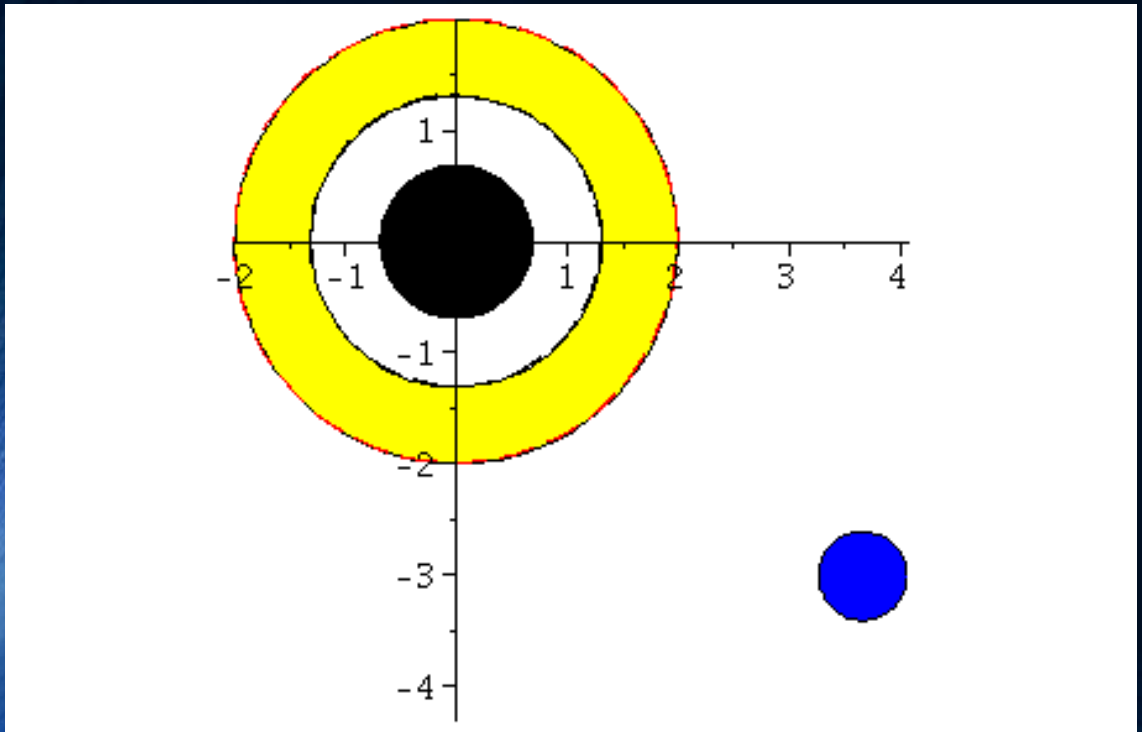


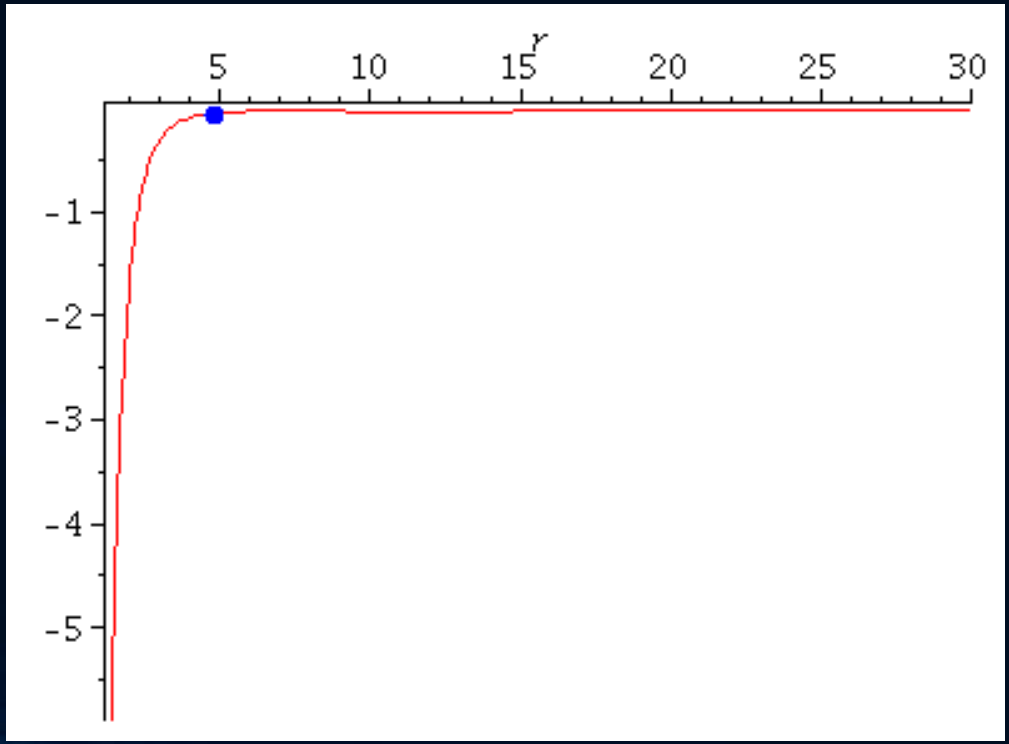
Probekörper rotiert entgegen der Rotationsrichtung des schwarzen Loches



Kerr Metrik:
Bewegungen innerhalb der Ergosphäre

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





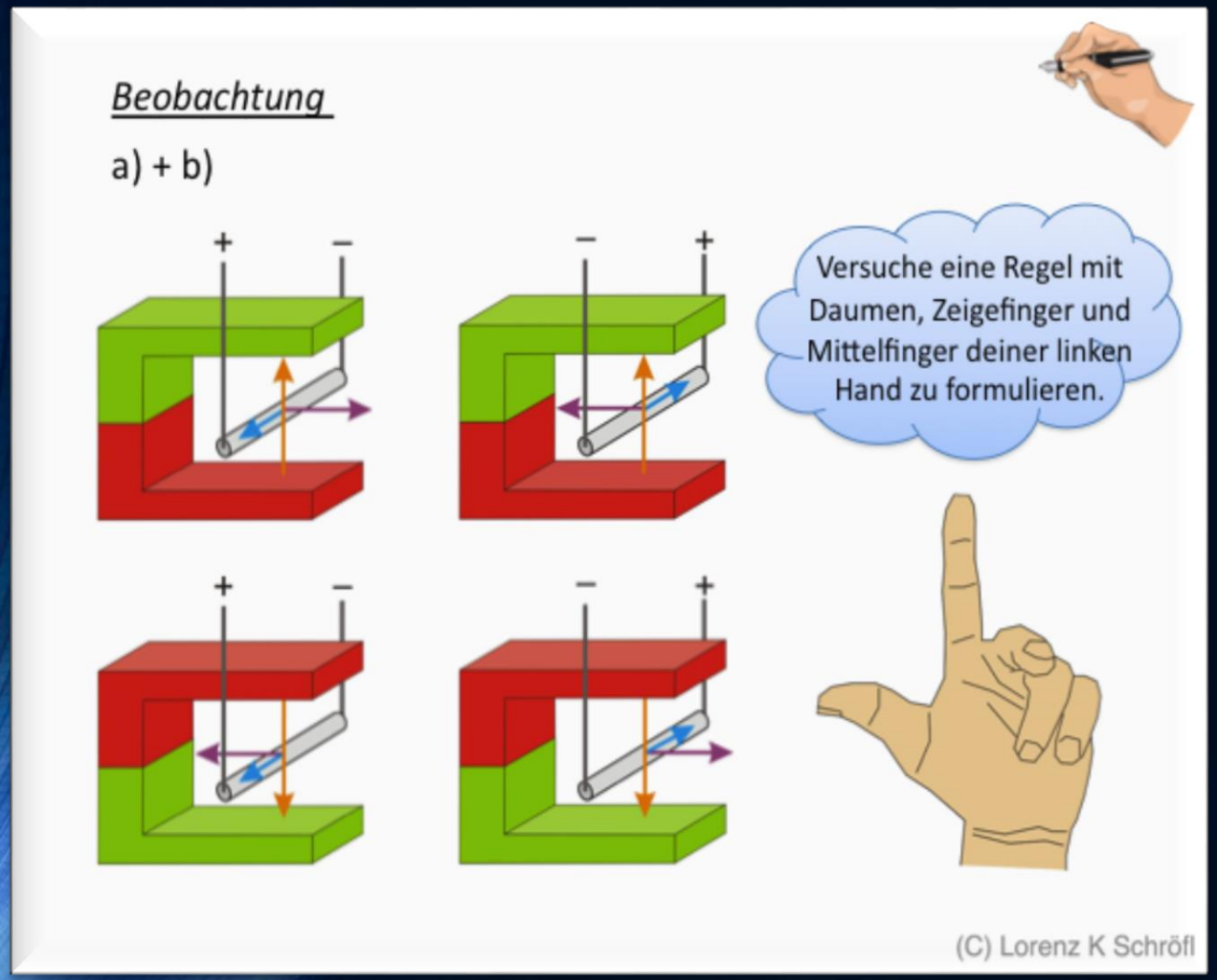
Der Mitführungseffekt der Raumzeit (“Frame-Dragging”)

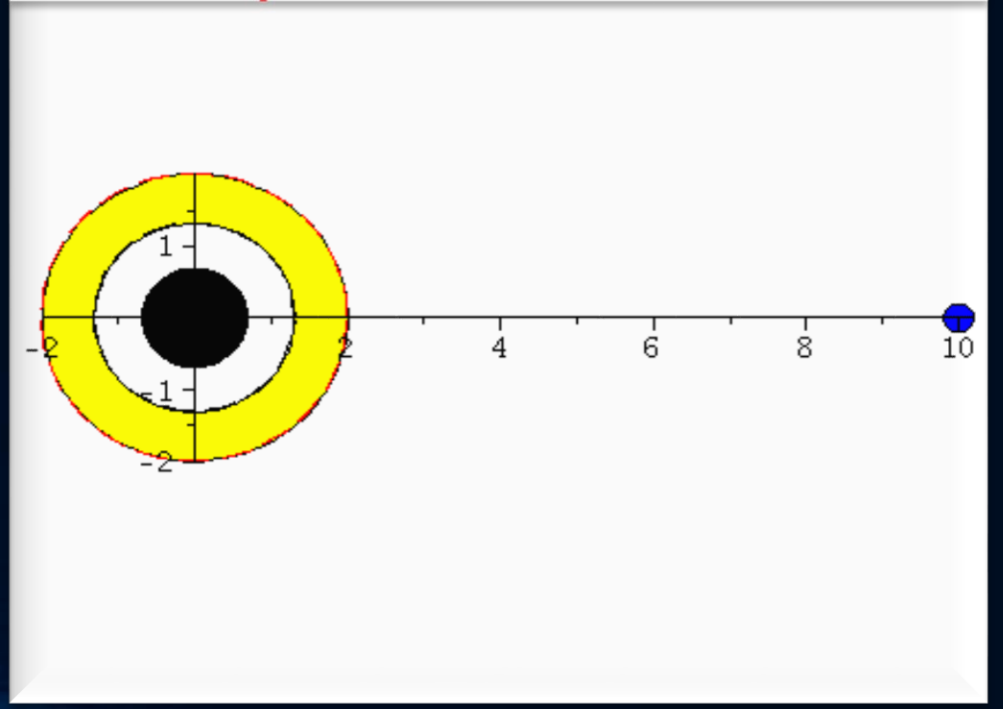
Experimente zur Bestätigung des Effektes: LARES, Gravity Probe B

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

Gravity Probe B

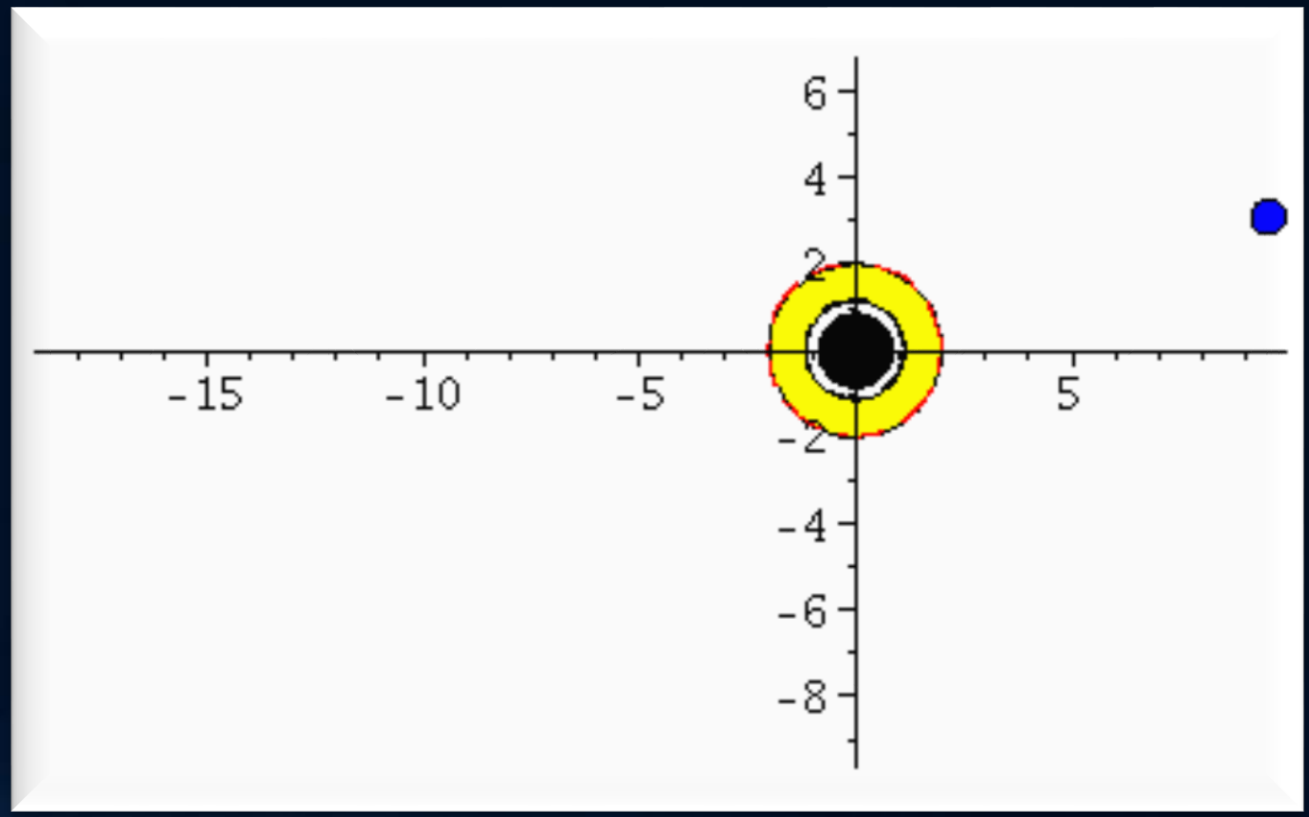
Der gravitomagnetische Effekt



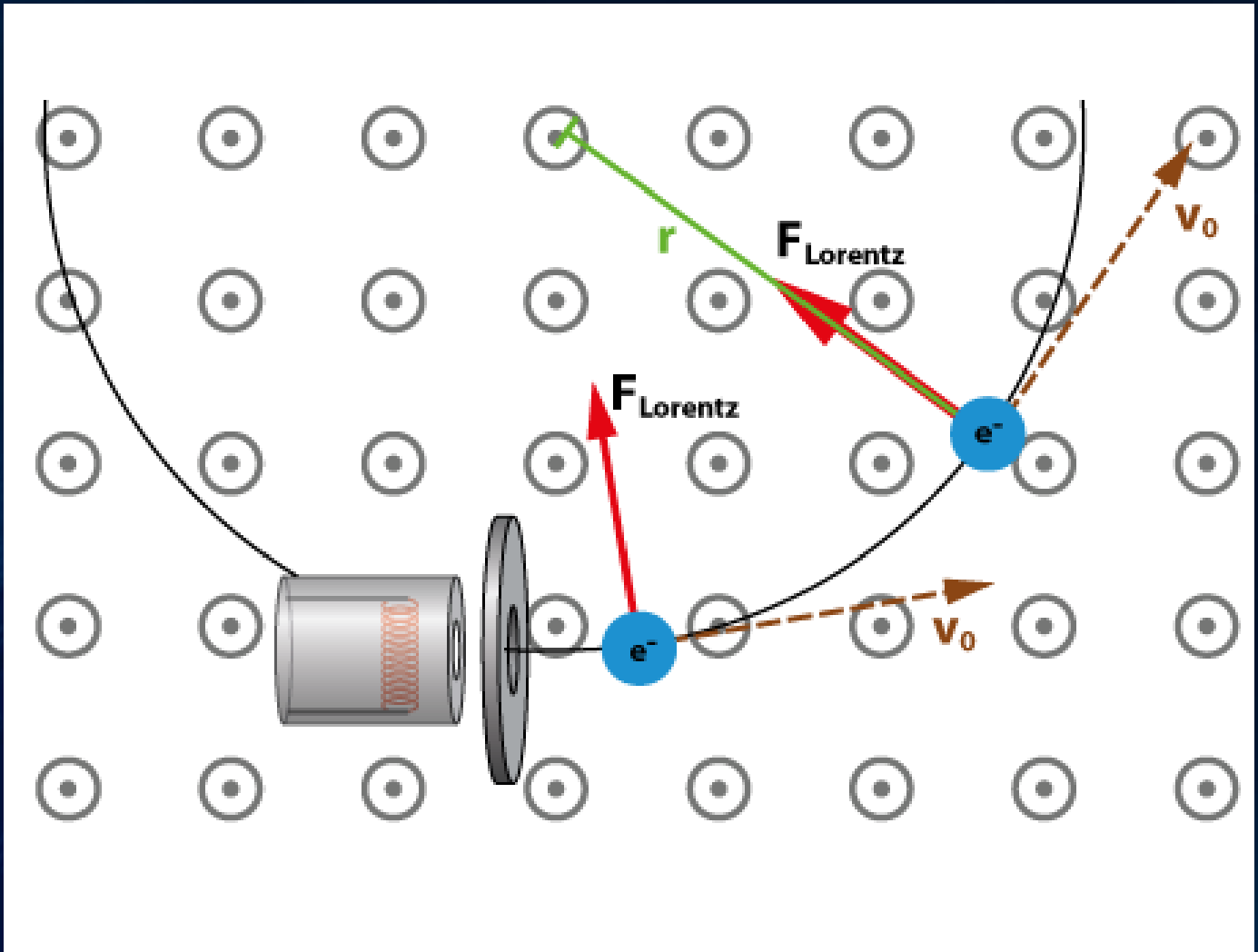


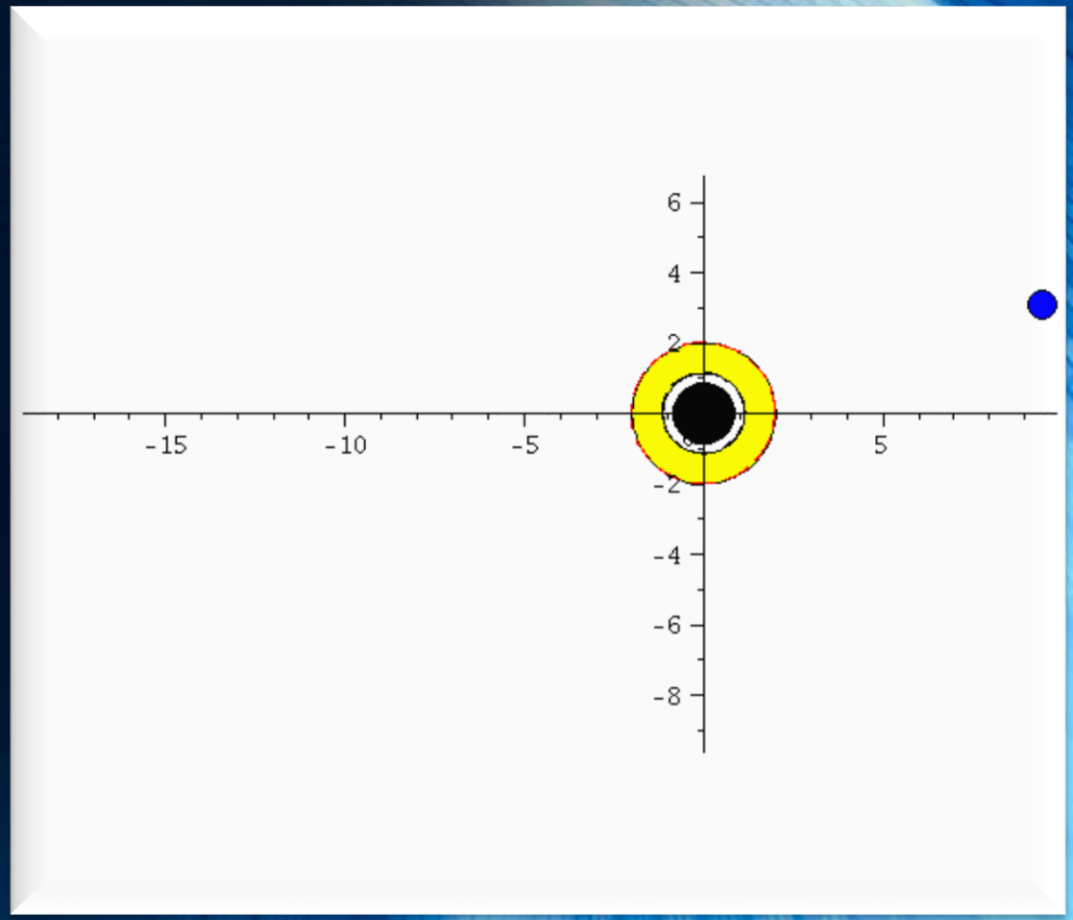
Kerr Metrik: Der gravitomagnetische Effekt

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).



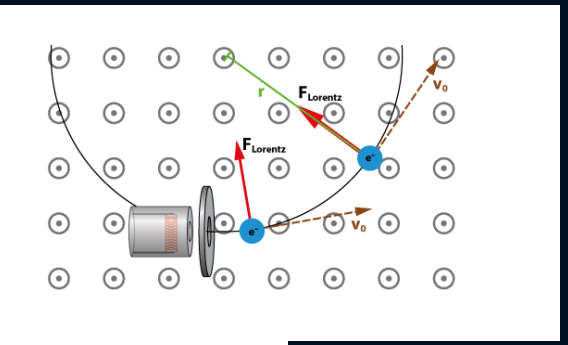
Der gravitomagnetische Effekt





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:

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

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.

C++ Grundgerüst und Variablen

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

Beginn des Hauptprogramms

Ausgabe eines Strings

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

Variablen bekommen einen festen Zahlenwert (Initialisierung)

Ausgabe des Wertes der Variablen

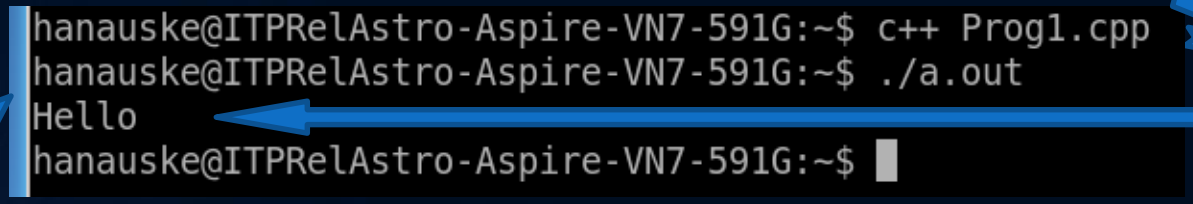
```
#include <iostream>                    //Ein-/Ausgabe (Include-Dateien)
using namespace std;                   //Fuer cout

main(void)  //Hauptprogramm
  {

    cout<<"Hello \n";

  }
```


```
#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



Das Programm wird gestartet und erzeugt im Terminal die Ausgabe “Hello”

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

./a.out

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;).

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

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:

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

TOV-Gleichungen: Numerisches Vorgehen

- 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.

C++ Lösen der TOV-Gleichung

```cpp
#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