Einführung in die Programmierung für Studierende der Physik

JOHANN WOLFGANG GOETHE UNIVERSITÄT
14.07.2022

MATTHIAS HANAUSKE

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

12. Vorlesung

Plan für die heutige Vorlesung

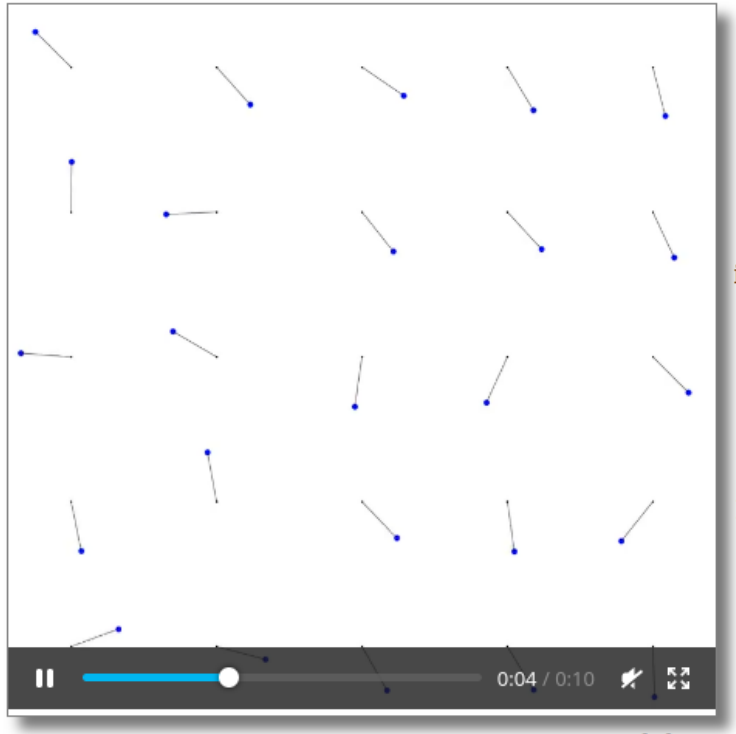
- Das Paradigma der parallelen Programmierung und das Rechnen auf Supercomputern
- Anwendungsbeispiele und weiterführende Vorlesungen

Das Paradigma der parallelen Programmierung und das Rechnen auf Supercomputern

Vorlesung 12

In dieser Vorlesung wird zunächst auf das *Paradigma der parallelen Programmierung und das Rechnen auf Supercomputern* eingegangen. Bei aufwendigen und rechenintensiven großen Programmen ist es oft nötig, die Programme zu parallelisieren; d.h. den Quelltext des Programms so umzuschreiben, dass einzelne, voneinander unabhängige Teil-Tasks gleichzeitig nebeneinander ausgeführt werden können. Im zweiten Punkt werden zwei aktuelle, komplexe Anwendungsbeispiele vorgestellt, bei denen die theoretisch, physikalische Forschung auf Computersimulationen angewiesen ist.

Das Paradigma der parallelen Programmierung und das Rechnen auf Supercomputern



Computersimulationen von realistischen, komplizierten Problemen erfordern sogar auf *Supercomputern* (Hochleistungs-Großrechenanlagen, bei denen eine große Anzahl von Mehrprozessor-Computern zu einem Computer-Cluster System verbunden sind) oft eine enorme Rechenzeit. Bei der Konzeption der Simulationsprogramme ist es deshalb erforderlich, dass die Rechenleistung des Computers stets voll ausgelastet ist und separate, voneinander unabhängige Teilaufgaben innerhalb der Programme gleichzeitig (parallel) berechnet werden. Das *Paradigma der parallelen Programmierung* ist die Art und Weise, wie man ein Programm gestaltet, damit die in dem Programm enthaltenen Teilberechnungen (Tasks) möglichst gleichzeitig, nebeneinander berechnet werden können. Diese *Nebenläufigkeit* der Programmstränge, die gleichzeitige Ausführung mehrerer Tasks, lässt sich in vielen Programmiersprachen (nicht nur C++ und Python) relativ einfach mittels OpenMP (Open Multi-Processing) parallelisieren. Das in der vorigen Vorlesung dargestellte Programm "Eine Kiste voller Pendel" (siehe Vorlesung 11, Unterpunkt Klausurvorbereitung und prüfungsrelevante Themen) wird in diesem Unterpunkt verwendet, um das Prinzip der *parallelen Programmierung* zu erläutern. Näheres siehe Das Paradigma der parallelen Programmierung und das Rechnen auf Supercomputern

Vorlesung 12

Das Hauptanliegen dieser Vorlesung bestand darin, den Studierenden zu lehren, wie man eine numerische Lösung eines komplexen physikalischen Problems auf dem Computer erstellen kann. Studierende, die dabei Gefallen am Programmieren gefunden haben, empfehle ich (in eigener Sache) die folgenden weiterführenden Vorlesungen/Praktika:

- Physik der sozio-ökonomischen Systeme mit dem Computer, WS 2022/23
- Vorlesung Allgemeine Relativitätstheorie mit dem Computer, voraussichtlich SS 2023
- Computerexperimente im Physikalischen Praktikum für Fortgeschrittene. Am Institut für Theoretische Physik werden die Versuche "Radial Oscillations of a Relativistic Spherical Star" und "Collapse of an unstable Neutron Star to a Black Hole" von mir betreut (näheres siehe F+L-Praktikum am Institut für Theoretische Physik).

In den oben genannten Vorlesungen und Praktika werden zwei, thematisch sehr unterschiedliche, Systeme mittels des Computers analysiert. Die im nächsten Semester (WS 2022/23) stattfindende Vorlesung Physik der sozio-ökonomischen Systeme mit dem Computer ist ein stark interdisziplinäres Forschungsfeld, in dem die unterschiedlichen Sprachen der Soziologie, der Wirtschaftswissenschaften, der Biologie, der Physik und der Mathematik aufeinander treffen, um zu beschreiben und zu verstehen, wie sich Menschen untereinander verhalten. Der wissenschaftliche Untersuchungsgegenstand, das zu erforschende Ding, ist hierbei der Mensch und wir sind daran interessiert, wie sich dieser bei ökonomischen oder sozial relevanten strategischen Entscheidungen verhält. Wir sind demnach auf der Suche nach einer Art von elementarer Theorie des menschlichen Verhaltens, ähnlich der physikalischen Elementarteilchen-Theorie und ihrer fundamentalen Wechselwirkungen und die Theorie, welche wir zur Beschreibung des interdependenten Entscheidungsverhaltens benutzen werden, ist die sogenannte *Spieltheorie*. Agentenbasierte, auf spieltheoretischen Grundannahmen basierende Computersimulationen von komplexen sozio-ökonomischen

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.

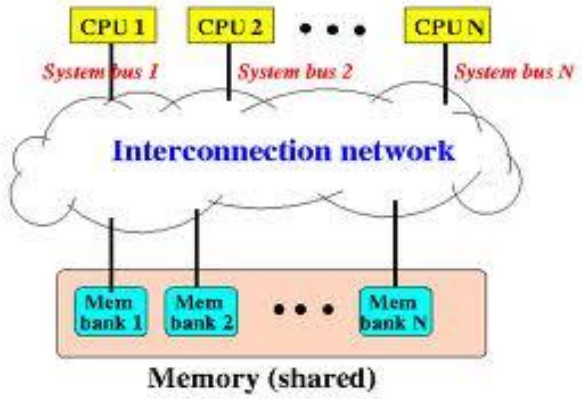
Depending on the architecture of the parallel computer (or computer cluster) the **Parallel Programming Framework** (OpenMP, MPI, Cuda, OpenCL, ...) has to be chosen .

Parallel computers

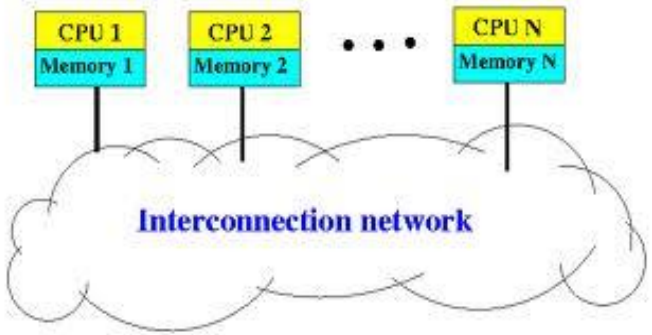
Parallel computer architectures:

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))

Das Paradigma der parallelen Programmierung und das Rechnen auf Supercomputern

Einführung

In diesem Unterpunkt werden wir uns mit dem Paradigma der parallelen Programmierung befassen. In diesem Programmierparadigma wird bei der Konzeption von rechenintensiven Computerprogrammen berücksichtigt, dass die Rechenleistung des Computers stets voll ausgelastet ist und separate, voneinander unabhängige Teilaufgaben (Tasks) innerhalb der Programme möglichst gleichzeitig (parallel) berechnet werden.

- **"Shared Memory" Mehrprozessorsysteme**:
  Mehrere Prozessoren eines Computers teilen sich einen gemeinsamen Arbeitsspeicher (RAM). Parallelisierung mittels OpenMP (Open Multi-Processing) relativ einfach möglich.
- **"Distributed Memory" Mehrprozessorsysteme**:
  Bei Hochleistungs-Großrechenanlagen sind mehrere Mehrprozessor-Computer zu Computer-Clustern verbunden, wobei jeder Computer seinen eigenen privaten Arbeitsspeicher hat. Parallelisierung mittels MPI (Message Passing Interface).

Die Art und Weise wie ein paralleles Programm zu konzipieren ist, hängt auch von der zugrundeliegenden Rechnerarchitektur des ausführenden Computers ab und man unterscheidet hier grob in die nebenstehenden Varianten. Viele Programmiersprachen lassen sich relativ einfach mittels OpenMP (Open Multi-Processing) parallelisieren, welches sich jedoch nur auf "Shared Memory" Systeme anwenden lässt. MPI (Message Passing Interface) stellt dagegen eine weit verbreitete Parallelisierungsmöglichkeit dar, die sowohl auf "Shared Memory" als auch auf "Distributed Memory" Systemen anwendbar ist. Die Parallelisierung eines sequentiellen Programms mittels MPI ist jedoch ein wenig aufwendiger. Dieser Unterpunkt gibt eine Einführung in die parallele Programmierung mit C++ und OpenMP. Die Parallelisierung eines Python-Programms ist aber natürlich auch möglich und kann z.B. mittels des Python-Moduls (threading oder multiprocessing) implementiert werden, bzw. unter Zuhilfenahme von MPI for Python realisiert werden.

Performance

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

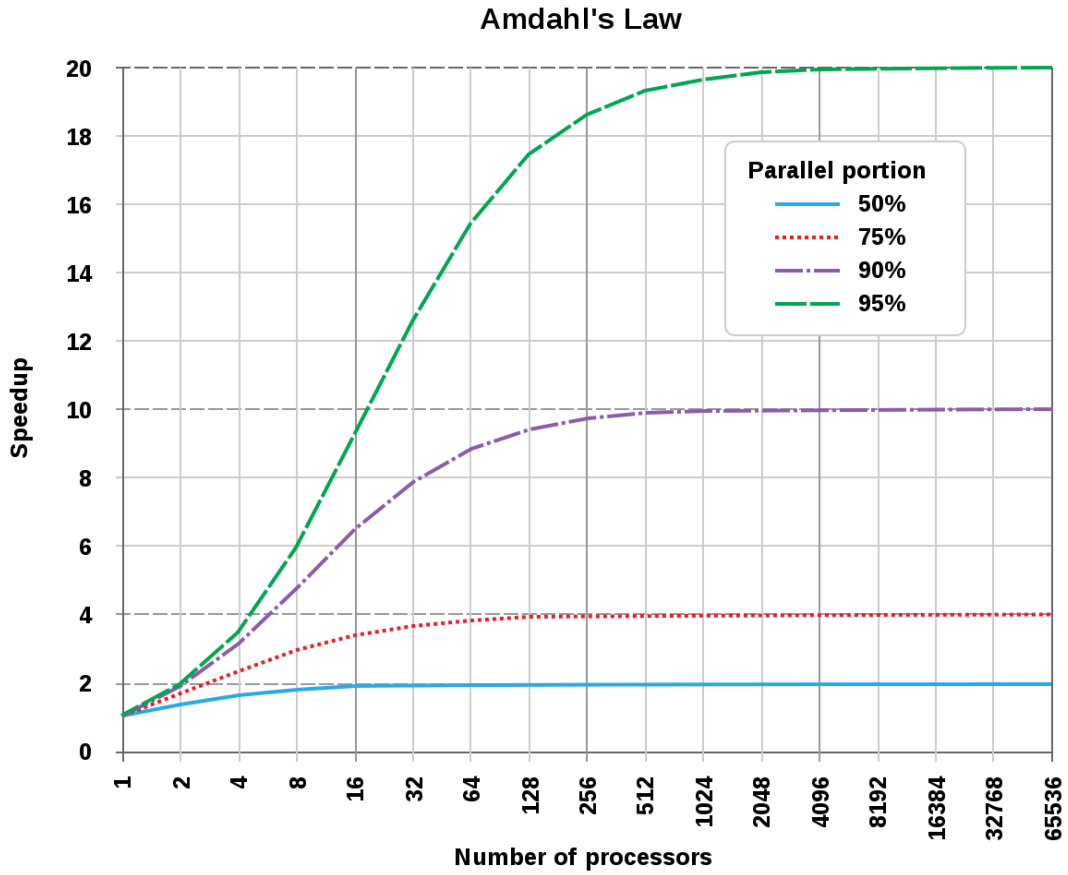
T(n): Time needed to run the program on n processing elements (e.g. CPU's, computer nodes).

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 divided in two parts (T(n)=Ts+Tp(n)), where Ts is the time needed for the non-parallelized part of the program and Tp(n) is the parallelized part, which can be executed concurrently by different processors.
A(n):=Max(S(n))

A(n)=1/(a+(1-a)/n), with 'Parallel portion' a:=Ts/T(1)



Beispiel: Parallelisierung der numerischen Berechnung der unendlich geometrischen Reihe

Parallelen Programmierung mit OpenMP

Die parallele Erweiterung eines sequentiellen C++ Programms (ein Programm ohne Parallelisierung) ist mittels OpenMP relativ einfach möglich. Betrachten wir z.B. das folgende C++ Programm, welches für neun unterschiedliche q-Werte den Wert der unendlich geometrischen Reihe berechnet (siehe Übungsblatt Nr. 3, Aufgabe 1). Um die Berechnung möglichst genau zu machen, wurde hierbei ein sehr hoher Wert (50 Millionen) als Obergrenze der Partialsumme benutzt

$$\sum_{k=0}^{50000000} q^k \approx \sum_{k=0}^{\infty} q^k = \frac{1}{1-q}, \quad \text{mit:} \quad q \in \mathbb{R} \ , \ |q| < 1 \quad .$$

Sequentielles (nicht-parallelisiertes Programm)
unendlich geometrischen Reihe

Geom_Reihe_sequentiell.cpp

```
/* Konvergenzverhalten der geometrischen Reihe für unterschiedliche q
 * Sequentielle Version der Berechnung von 9 approximierten Werten */
#include <iostream>                    // Ein- und Ausgabebibliothek
#include <cmath>                       // Bibliothek für mathematisches (e-Funktion, Betrag, ...)
using namespace std;                   // Benutze den Namensraum std

int main(){                                    // Hauptfunktion
    int startTime = time(NULL);                // Starten der Zeitmessung
    double q;                                  // Deklaration des Parameters q
    const unsigned long int N = 50000000; // Definition der Anzahl der mitgenommenen Summenglieder
    double wert_approx;                        // Deklaration des approximierten Wertes der Summe

    for(int i=1; i <= 9; ++i){         // Schleife zur ueber neun q-Werte
        q = i*0.1;                     // Festlegung des q-Wertes
        wert_approx = 0;               // Summenvariable auf Null setzen

        for(int k=0; k<=N; ++k){                    // Schleife zur Berechnung der Summe
            wert_approx = wert_approx + pow(q,k); // Innerer Teil der geometrischen Reihe
        }                                           // Ende der Schleife der Summenberechnung

        printf("q=%6.4f , Wert der Summe = ",q);  // Terminalausgabe der berechneten Werte
        printf("%13.10f (Fehler zum wirklichen Wert : %13.6e ) \n",wert_approx, wert_approx - 1.0/(1-q));
    }                                          // Ende der for-Schleife ueber die unterschiedlichen q-Werte
    // Terminalausgabe der benoetigten Zeit
    cout << "Das Programm benoetigte: " << time(NULL)-startTime << " Sekunden." << endl;
}   // Ende Hauptfunktion
```

Geom_Reihe_parallel.cpp

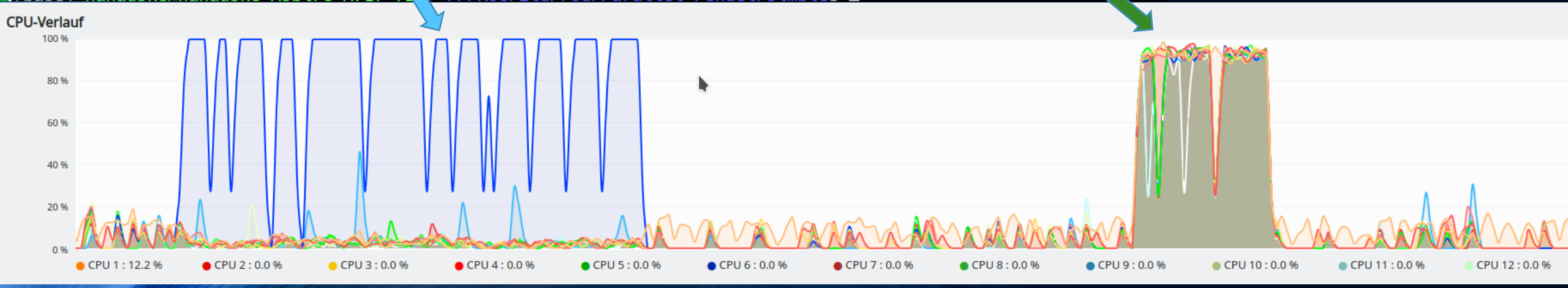
Paralleles Programm (1. Version)
unendlich geometrischen Reihe

```
/* Konvergenzverhalten der geometrischen Reihe für unterschiedliche q
 * Parallele Version der Berechnung von 9 approximierten Werten
 * Parallelisierung in der for-Schleife ueber neun q-Werte
 */
#include <iostream>                  // Ein- und Ausgabebibliothek
#include <cmath>                     // Bibliothek für mathematisches (e-Funktion, Betrag, ...)
#include <omp.h>                     // OpenMP zum parallelen Rechnen
using namespace std;                 // Benutze den Namensraum std

int main(){                                  // Hauptfunktion
    int startTime = time(NULL);              // Starten der Zeitmessung
    double q;                                // Deklaration des Parameters q
    const unsigned long int N = 50000000; // Definition der Anzahl der mitgenommenen Summenglieder
    double wert_approx;                      // Deklaration des approximierten Wertes der Summe

    #pragma omp parallel for private(q,wert_approx)
    for(int i=1; i <= 9; ++i){           // Schleife zur ueber neun q-Werte
        q = i*0.1;                       // Festlegung des q-Wertes
        wert_approx = 0;                 // Summenvariable auf Null setzen

        for(int k=0; k<=N; ++k){                    // Schleife zur Berechnung der Summe
            wert_approx = wert_approx + pow(q,k); // Innerer Teil der geometrischen Reihe
        }                                           // Ende der Schleife der Summenberechnung

        printf("q=%6.4f , Wert der Summe = ",q);  // Terminalausgabe der berechneten Werte
        printf("%13.10f (Fehler zum wirklichen Wert : %13.6e ) \n",wert_approx, wert_approx - 1.0/(1-q));
    }                                           // Ende der for-Schleife ueber die unterschiedlichen q-Werte
    // Terminalausgabe der benoetigten Zeit
    cout << "Das Programm benoetigte: " << time(NULL)-startTime << " Sekunden." << endl;
}   // Ende Hauptfunktion
```

Vergleich der Performance der beiden Programme

```
(base) hanauske@hanauske-Aspire-A717-72G:~/PPROG/EigProg/Parallel_Pendel/simple$ g++ -fopenmp Geom_Reihe_parallel.cpp
(base) hanauske@hanauske-Aspire-A717-72G:~/PPROG/EigProg/Parallel_Pendel/simple$ ./a.out
q=0.3000 , Wert der Summe =  1.4285714286 (Fehler zum wirklichen Wert :  2.220446e-16 )
q=0.1000 , Wert der Summe =  1.1111111111 (Fehler zum wirklichen Wert :  0.000000e+00 )
q=0.2000 , Wert der Summe =  1.2500000000 (Fehler zum wirklichen Wert :  0.000000e+00 )
q=0.7000 , Wert der Summe =  3.3333333333 (Fehler zum wirklichen Wert : -1.776357e-15 )
q=0.6000 , Wert der Summe =  2.5000000000 (Fehler zum wirklichen Wert :  0.000000e+00 )
q=0.5000 , Wert der Summe =  2.0000000000 (Fehler zum wirklichen Wert :  0.000000e+00 )
q=0.4000 , Wert der Summe =  1.6666666667 (Fehler zum wirklichen Wert :  4.440892e-16 )
q=0.8000 , Wert der Summe =  5.0000000000 (Fehler zum wirklichen Wert :  0.000000e+00 )
q=0.9000 , Wert der Summe = 10.0000000000 (Fehler zum wirklichen Wert : -8.881784e-15 )
Das Programm benoetigte: 5 Sekunden.
```

```
(base) hanauske@hanauske-Aspire-A717-72G:~/PPROG/EigProg/Parallel_Pendel/simple$ g++ Geom_Reihe_sequentiell.cpp
(base) hanauske@hanauske-Aspire-A717-72G:~/PPROG/EigProg/Parallel_Pendel/simple$ ./a.out
q=0.1000 , Wert der Summe =  1.1111111111 (Fehler zum wirklichen Wert :  0.000000e+00 )
q=0.2000 , Wert der Summe =  1.2500000000 (Fehler zum wirklichen Wert :  0.000000e+00 )
q=0.3000 , Wert der Summe =  1.4285714286 (Fehler zum wirklichen Wert :  2.220446e-16 )
q=0.4000 , Wert der Summe =  1.6666666667 (Fehler zum wirklichen Wert :  4.440892e-16 )
q=0.5000 , Wert der Summe =  2.0000000000 (Fehler zum wirklichen Wert :  0.000000e+00 )
q=0.6000 , Wert der Summe =  2.5000000000 (Fehler zum wirklichen Wert :  0.000000e+00 )
q=0.7000 , Wert der Summe =  3.3333333333 (Fehler zum wirklichen Wert : -1.776357e-15 )
q=0.8000 , Wert der Summe =  5.0000000000 (Fehler zum wirklichen Wert :  0.000000e+00 )
q=0.9000 , Wert der Summe = 10.0000000000 (Fehler zum wirklichen Wert : -8.881784e-15 )
Das Programm benoetigte: 28 Sekunden.
(base) hanauske@hanauske-Aspire-A717-72G:~/PPROG/EigProg/Parallel_Pendel/simple$
```



Paralleles Programm (2. Version)
unendlich geometrischen Reihe

Geom_Reihe_parallel_1.cpp

```cpp
/* Konvergenzverhalten der geometrischen Reihe für unterschiedliche q
 * Parallele Version der Berechnung von 9 approximierten Werten
 * Parallelisierung in der for-Schleife zur Berechnung der Summe bei festem q
 */
#include <iostream>                    // Ein- und Ausgabebibliothek
#include <cmath>                       // B
#include <omp.h>                       // O
using namespace std;                   // B

int main(){
    int startTime = time(NULL);
    double q;
    const unsigned long int N = 50000000;
    double wert_approx;

    for(int i=1; i <= 9; ++i){         // Schleife zur ueber neun q-Werte
        q = i*0.1;                     // Festlegung des q-Wertes
        wert_approx = 0;               // Summenvariable auf Null setzen

        #pragma omp parallel for reduction(+:wert_approx)
        for(int k=0; k<=N; ++k){                    // Schleife zur Berechnung der Summe
            wert_approx = wert_approx + pow(q,k); // Innerer Teil der geometrischen Reihe
        }                                           // Ende der Schleife der Summenberechnung

        printf("q=%6.4f , Wert der Summe = ",q);  // Terminalausgabe der berechneten Werte
        printf("%13.10f (Fehler zum wirklichen Wert : %13.6e ) \n",wert_approx, wert_approx - 1.0/(1-q));
    }                                               // Ende der for-Schleife ueber die unterschiedlichen q-Werte
    // Terminalausgabe der benoetigten Zeit
    cout << "Das Programm benoetigte: " << time(NULL)-startTime << " Sekunden." << endl;
}   // Ende Hauptfunktion
```

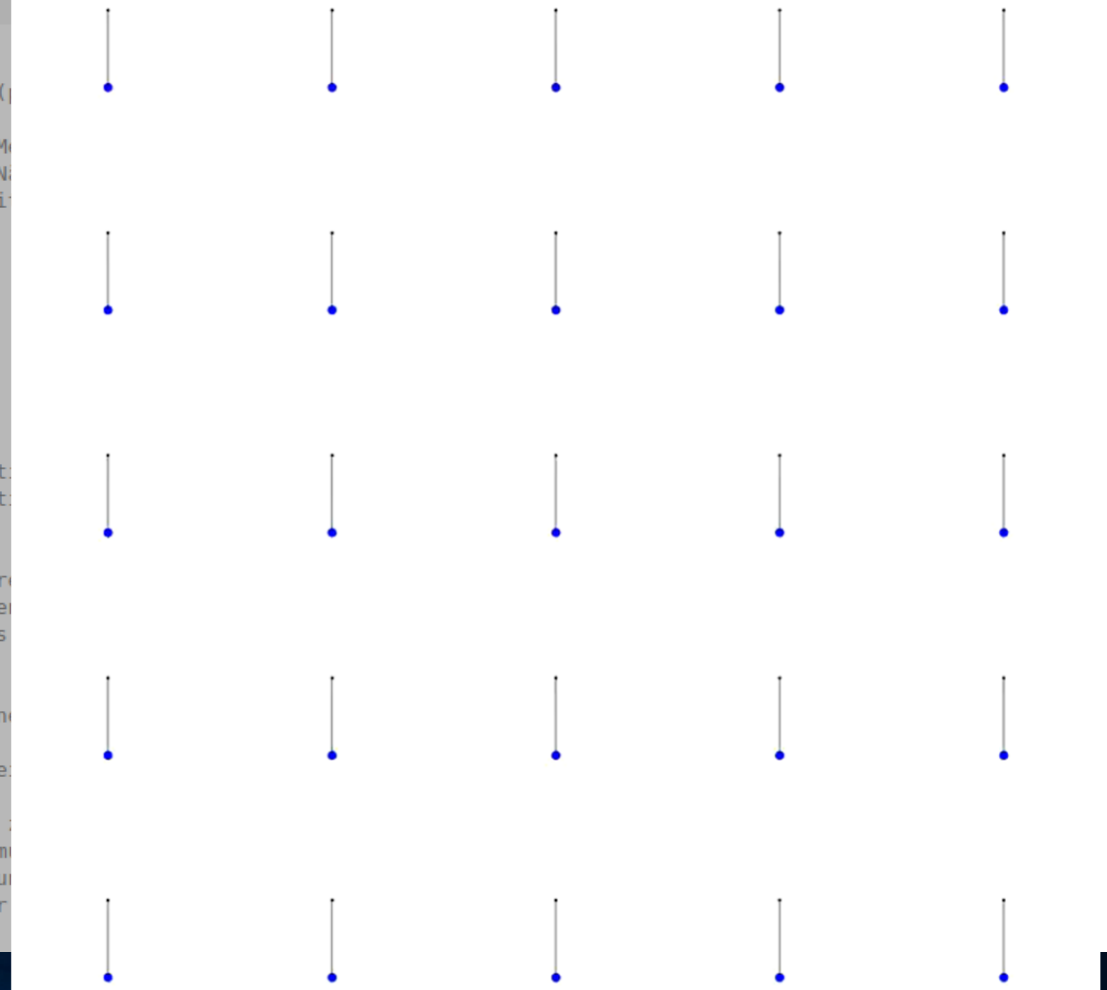
```
(base) hanauske@hanauske-Aspire-A717-72G:~/www/Versuch1/VPROG/C++$ g++ -fopenmp Geom_Reihe_parallel_1.cpp
(base) hanauske@hanauske-Aspire-A717-72G:~/www/Versuch1/VPROG/C++$ ./a.out
q=0.1000 , Wert der Summe =  1.1111111111 (Fehler zum wirklichen Wert :  0.000000e+00 )
q=0.2000 , Wert der Summe =  1.2500000000 (Fehler zum wirklichen Wert :  0.000000e+00 )
q=0.3000 , Wert der Summe =  1.4285714286 (Fehler zum wirklichen Wert :  2.220446e-16 )
q=0.4000 , Wert der Summe =  1.6666666667 (Fehler zum wirklichen Wert :  4.440892e-16 )
q=0.5000 , Wert der Summe =  2.0000000000 (Fehler zum wirklichen Wert :  0.000000e+00 )
q=0.6000 , Wert der Summe =  2.5000000000 (Fehler zum wirklichen Wert :  0.000000e+00 )
q=0.7000 , Wert der Summe =  3.3333333333 (Fehler zum wirklichen Wert : -1.776357e-15 )
q=0.8000 , Wert der Summe =  5.0000000000 (Fehler zum wirklichen Wert :  0.000000e+00 )
q=0.9000 , Wert der Summe = 10.0000000000 (Fehler zum wirklichen Wert : -8.881784e-15 )
Das Programm benoetigte: 5 Sekunden.
```

Beispiel: Die parallele OpenMP-Version des C++ Programms "Eine Kiste voller Pendel"

Im Folgenden werden wir eine parallele OpenMP-Version des C++ Programms "Eine Kiste voller Pendel" (siehe Vorlesung 11, Unterpunkt Klausurvorbereitung und prüfungsrelevante Themen) erstellen. Wir benutzen dafür die folgende sequentielle Version des C++ Programms, wobei wir im Vergleich zur ursprünglichen Version zusätzlich eine Zeitmessung eingebaut haben und die Anzahl der Zeitgitterpunkte und das zu simulierende Zeitintervall erhöht haben:

Pendel_Container_sequentiell.cpp

```
/* Sequenzielle Version
 * Beispiel einer, von der Basisklasse 'Ding' abgeleitete Klasse 'Pendel'
 * Zwei zusaetzliche private Instanzvariablen (l und beta) und die winkelspezifischen Anfangswerte (
 * wurden der abgeleiteten Pendel-Klasse hinzugefuegt
 * Der Konstruktor der Klasse Pendel erzeugt ein Objekt Ding und initialisiert seine eigenen Daten-M
 * Von der Klasse Pendel wurde eine weitere Subklasse 'Pendel_math' abgeleitet, welche die lineare N
 * Direkte Ausgabe der berechneten Werte von 25 Pendel-Simulationen in eine Datei (Visualisierung mi
 * Zusatzliche Zeitmessung zum späteren Vergleich mit der parallelen Variante
*/
#include "Pendel.hpp"          // Pendel und Ding Klassen
#include <iostream>            // Ein- und Ausgabebibliothek
#include <vector>              // Sequenzieller Container vector<Type> der Standardbibliothek

int main(){                     // Hauptfunktion
    int startTime = time(NULL); // Starten der Zeitmessung
    const int Anz_Pendel = 25;  // Anzahl der zu berechnenden Pendel-Simulationen
    double v_phi_0 = 8.2;       // Anfangsgeschwindigkeitsbereich (kleinster Wert) der Pendelsimulat
    double beta = 0.0;          // Anfangsgeschwindigkeitsbereich (kleinster Wert) der Pendelsimulat
    double a = 0.0;             // Untergrenze des Zeit-Intervalls
    double b = 50.0;            // Obergrenze des Zeit-Intervalls
    int N_RK = 20000;           // Anzahl der Gitter-Zeitpunkte des Runge-Kutta Ordnung vier Verfahr
    unsigned int N_sim = 2000;  // Anzahl der Gitter-Zeitpunkte der Simulation (Anzahl der ausgegebe
    double dt = (b - a)/N_sim;  // Abstand dt zwischen den aequidistanten Punkten des Sim-Intervalls

    FILE *ausgabe;                                  // Deklaration eines Files fuer die Ausgabedatei
    ausgabe = fopen("Pendel_Container.dat", "w+"); // Ergebnisse werden in die Datei "Pendel_Contain

    vector<Pendel> Kiste_Pendel;                    // Deklaration eines vector-Containers fuer die e

    for (unsigned int n = 0; n < Anz_Pendel; ++n){                                  // for-Schleife
        if ((n % 5) == 0) { v_phi_0 = v_phi_0 * ( 1 + double(n)/55 ); beta = 0.0; } // Jeweils 5 Sim
        Kiste_Pendel.push_back( Pendel {a, 0.0, v_phi_0, 0.5, beta, N_sim} );       // Initialisieru
        beta = beta + 0.2;                                                          // Der Parameter
    }
```



Pendel_Container_sequentiell.cpp

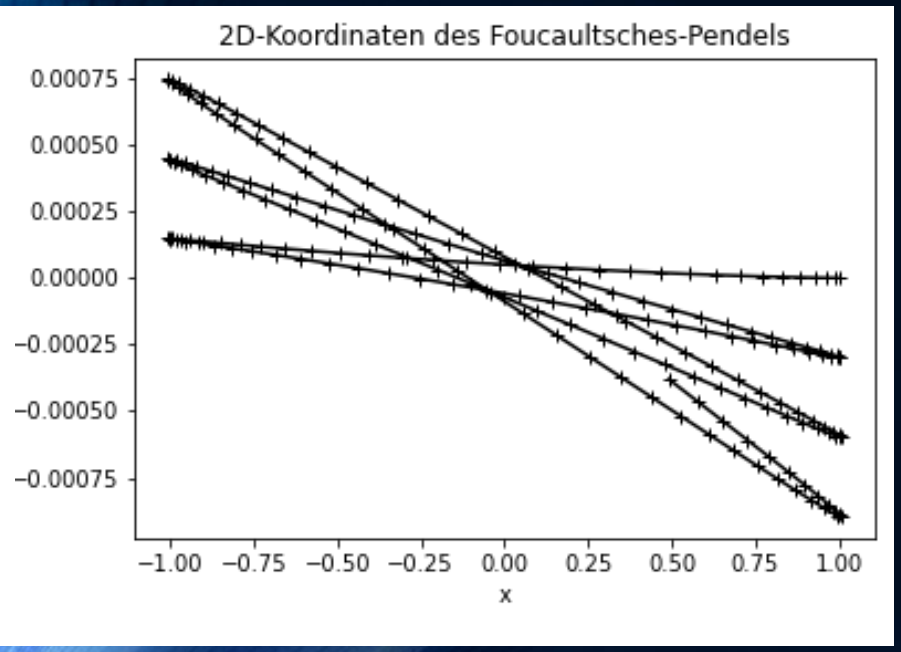
```cpp
/* Sequenzielle Version
 * Beispiel einer, von der Basisklasse 'Ding' abgeleitete Klasse 'Pendel'
 * Zwei zusaetzliche private Instanzvariablen (l und beta) und die winkelspezifischen Anfangswerte (phi und v_phi)
 * wurden der abgeleiteten Pendel-Klasse hinzugefuegt
 * Der Konstruktor der Klasse Pendel erzeugt ein Objekt Ding und initialisiert seine eigenen Daten-Member
 * Von der Klasse Pendel wurde eine weitere Subklasse 'Pendel_math' abgeleitet, welche die lineare Näherung der Pendel-DGL benutzt (gedaempfter harmonischer Ozillator)
 * Direkte Ausgabe der berechneten Werte von 25 Pendel-Simulationen in eine Datei (Visualisierung mittels Pendel_Container.py)
 * Zusatzliche Zeitmessung zum späteren Vergleich mit der parallelen Variante
*/
#include "Pendel.hpp"          // Pendel und Ding Klassen
#include <iostream>            // Ein- und Ausgabebibliothek
#include <vector>              // Sequenzieller Container vector<Type> der Standardbibliothek

int main(){                    // Hauptfunktion
    int startTime = time(NULL); // Starten der Zeitmessung
    const int Anz_Pendel = 25;  // Anzahl der zu berechnenden Pendel-Simulationen
    double v_phi_0 = 8.2;       // Anfangsgeschwindigkeitsbereich (kleinster Wert) der Pendelsimulationen
    double beta = 0.0;          // Anfangsgeschwindigkeitsbereich (kleinster Wert) der Pendelsimulationen
    double a = 0.0;             // Untergrenze des Zeit-Intervalls
    double b = 50.0;            // Obergrenze des Zeit-Intervalls
    int N_RK = 20000;           // Anzahl der Gitter-Zeitpunkte des Runge-Kutta Ordnung vier Verfahrens
    unsigned int N_sim = 2000;  // Anzahl der Gitter-Zeitpunkte der Simulation (Anzahl der ausgegebenen Punkte)
    double dt = (b - a)/N_sim;  // Abstand dt zwischen den aequidistanten Punkten des Sim-Intervalls (h=dt)

    FILE *ausgabe;                                  // Deklaration eines Files fuer die Ausgabedatei
    ausgabe = fopen("Pendel_Container.dat", "w+"); // Ergebnisse werden in die Datei "Pendel_Container.dat" geschrieben

    vector<Pendel> Kiste_Pendel;                    // Deklaration eines vector-Containers fuer die einzelnen Pendel-Objekte

    for (unsigned int n = 0; n < Anz_Pendel; ++n){                                      // for-Schleife zum Auffuellen des Containers mit Elementen vom Typ 'Pendel'
        if ((n % 5) == 0) { v_phi_0 = v_phi_0 * ( 1 + double(n)/55 ); beta = 0.0; } // Jeweils 5 Simulationen werden mit gleicher Anfangsgeschwindigkeit gemacht
        Kiste_Pendel.push_back( Pendel {a, 0.0, v_phi_0, 0.5, beta, N_sim} );       // Initialisierung der Pendel-Objekte
        beta = beta + 0.2;                                                          // Der Parameter beta der Reibung wird veraendert
    }

    // Beschreibung der in die Ausgabedatei "Pendel.dat" ausgegebenen Groessen
    fprintf(ausgabe, "%10s %20s %20s %20s %20s %20s %20s %20s %20s \n", "# Index i", "t-Wert", "x(t), Pendel 1", "y(t), Pendel 1", "v_x(t), Pendel 1", "v_y(t), Pendel 1", "x(t), Pendel 2", "...");

    // Zeitliche Simulation
    for(int i=0; i  <= N_sim; ++i){                             // for-Schleife  ueber die Zeitgitterpunkte der Simulation
        fprintf(ausgabe, "%10d %20.12f ",i,  Kiste_Pendel[0].t); // Ausgabe des Zeit-Indexes und der aktuellen Zeit in die Ausgabedatei
        for (auto& n : Kiste_Pendel){                           // Bereichsbasierte for-Schleife ueber die einzelnen Pendel-Objekte
            fprintf(ausgabe, "%20.12f %20.12f %20.12f %20.12f ",n.r[0], n.r[1], n.v[0], n.v[1]); // Ausgabe der Orts- und Geschwindigkeitswerte des Pendel-Objektes in die Ausgabedatei
            n.Gehe_Zeitschritt(dt, N_RK);                       // Aufruf der Funktion Gehe_Zeitschritt(...)
        }                                                       // Ende for-Schleife (Pendel-Objekte)
        fprintf(ausgabe, "\n");                                 // Neue Zeile in der Ausgabedatei
    }                                                           // Ende for-Schleife (Zeitgitterpunkte)
    fclose(ausgabe);                                            // Schliessen der Ausgabedatei
    cout << "Das Programm benoetigte: " << time(NULL)-startTime << " Sekunden." << endl; // Terminalausgabe der benoetigten Zeit
}                   // Ende main()-Funktion
```

Pendel_Container_parallel.cpp

```cpp
/* Sequenzielle Version
 * Beispiel einer, von der Basisklasse 'Ding' abgeleitete Klasse 'Pendel'
 * Zwei zusaetzliche private Instanzvariablen (l und beta) und die winkelspezifischen Anfangswerte (phi und v_phi)
 * wurden der abgeleiteten Pendel-Klasse hinzugefuegt
 * Der Konstruktor der Klasse Pendel erzeugt ein Objekt Ding und initialisiert seine eigenen Daten-Member
 * Von der Klasse Pendel wurde eine weitere Subklasse 'Pendel_math' abgeleitet, welche die lineare Näherung der Pendel-DGL benutzt (gedaempfter harmonischer Ozillator)
 * Direkte Ausgabe der berechneten Werte von 25 Pendel-Simulationen in eine Datei (Visualisierung mittels Pendel_Container.py)
 * Zusatzliche Zeitmessung zum späteren Vergleich mit der parallelen Variante
*/
#include "Pendel.hpp"           // Pendel und Ding Klassen
#include <iostream>             // Ein- und Ausgabebibliothek
#include <vector>               // Sequenzieller Container vector<Type> der Standardbibliothek
#include <omp.h>                // OpenMP zum parallelen Rechnen

int main(){                     // Hauptfunktion
    int startTime = time(NULL); // Starten der Zeitmessung
    const int Anz_Pendel = 25;  // Anzahl der zu berechnenden Pendel-Simulationen
    double v_phi_0 = 8.2;       // Anfangsgeschwindigkeitsbereich (kleinster Wert) der Pendelsimulationen
    double beta = 0.0;          // Anfangsgeschwindigkeitsbereich (kleinster Wert) der Pendelsimulationen
    double a = 0.0;             // Untergrenze des Zeit-Intervalls
    double b = 50.0;            // Obergrenze des Zeit-Intervalls
    int N_RK = 20000;           // Anzahl der Gitter-Zeitpunkte des Runge-Kutta Ordnung vier Verfahrens
    unsigned int N_sim = 2000;  // Anzahl der Gitter-Zeitpunkte der Simulation (Anzahl der ausgegebenen Punkte)
    double dt = (b - a)/N_sim;  // Abstand dt zwischen den aequidistanten Punkten des Sim-Intervalls (h=dt)

    FILE *ausgabe;                                  // Deklaration eines Files fuer die Ausgabedatei
    ausgabe = fopen("Pendel_Container.dat", "w+"); // Ergebnisse werden in die Datei "Pendel_Container.dat" geschrieben

    vector<Pendel> Kiste_Pendel;                    // Deklaration eines vector-Containers fuer die einzelnen Pendel-Objekte

    for (unsigned int n = 0; n < Anz_Pendel; ++n){                                  // for-Schleife zum Auffuellen des Containers mit Elementen vom Typ 'Pendel'
        if ((n % 5) == 0) { v_phi_0 = v_phi_0 * ( 1 + double(n)/55 ); beta = 0.0; } // Jeweils 5 Simulationen werden mit gleicher Anfangsgeschwindigkeit gemacht
        Kiste_Pendel.push_back( Pendel {a, 0.0, v_phi_0, 0.5, beta, N_sim} );       // Initialisierung der Pendel-Objekte
        beta = beta + 0.2;                                                          // Der Parameter beta der Reibung wird veraendert
    }

    // Beschreibung der in die Ausgabedatei "Pendel.dat" ausgegebenen Groessen
    fprintf(ausgabe, "%10s %20s %20s %20s %20s %20s %20s %20s %20s \n", "# Index i", "t-Wert", "x(t), Pendel 1", "y(t), Pendel 1", "v_x(t), Pendel 1", "v_y(t), Pendel 1", "x(t), Pendel 2", "...");

    // Zeitliche Simulation
    for(int i=0; i  <= N_sim; ++i){                             // for-Schleife  ueber die Zeitgitterpunkte der Simulation
        fprintf(ausgabe, "%10d %20.12f ",i,  Kiste_Pendel[0].t); // Ausgabe des Zeit-Indexes und der aktuellen Zeit in die Ausgabedatei
        for (int n=0; n < Kiste_Pendel.size(); ++n){            // for-Schleife ueber die einzelnen Pendel-Objekte
            fprintf(ausgabe, "%20.12f %20.12f %20.12f %20.12f ",Kiste_Pendel[n].r[0], Kiste_Pendel[n].r[1], Kiste_Pendel[n].v[0], Kiste_Pendel[n].v[1]);
        }                                                       // Ende for-Schleife (Pendel-Objekte)
        fprintf(ausgabe, "\n");                                 // Neue Zeile in der Ausgabedatei

        #pragma omp parallel for                                // Parallelisierungsanweisung fuer die folgende for-Schleife
        for (int n=0; n < Kiste_Pendel.size(); ++n){            // for-Schleife ueber die einzelnen Pendel-Objekte
            Kiste_Pendel[n].Gehe_Zeitschritt(dt, N_RK);         // Aufruf der Funktion Gehe_Zeitschritt(...)
        }                                                       // Ende for-Schleife (Pendel-Objekte)
    }                                                           // Ende for-Schleife (Zeitgitterpunkte)
    fclose(ausgabe);                                            // Schliessen der Ausgabedatei
    cout << "Das Programm benoetigte: " << time(NULL)-startTime << " Sekunden." << endl; // Terminalausgabe der benoetigten Zeit
}                   // Ende main() Funktion
```

Studentische Projekte

Das Foucaultsche Pendel



Rosette_mit_w=1_und_Anfangsgeschwindigkeit.py

Users > rominaghasemizadeh > Desktop > Rosette_mit_w=1_und_Anfangsgeschwindigkeit.py > ...

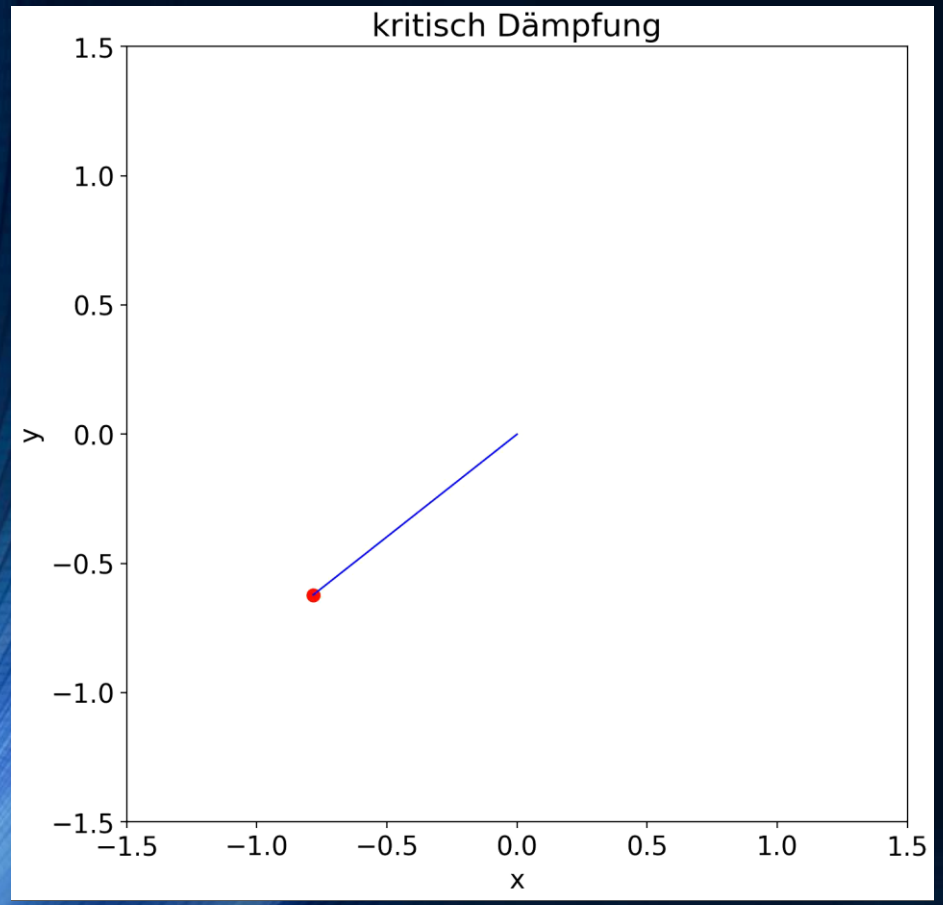
```python
import matplotlib.pyplot as plt
from matplotlib.animation import FuncAnimation
import numpy as np

data = np.genfromtxt("./Foucault_2dim_w_1_Anfangsgeschw.dat")
fig, vid = plt.subplots()
line1 = vid.plot(data[0][2],data[0][3],color="orange")[0]

def update(i):
    line1.set_data(data[:i*1000,2], data[:i*1000,3])
    print(i)
    return line1

anim = FuncAnimation(fig,update ,frames=1000000, interval=2, blit=False)
plt.xlim([-0.075,0.075])
plt.ylim([-0.075,0.075])
vid.set_title("Rosette mit w=1 und Anfangsgeschwindigkeit ")
vid.set_xlabel("X")
vid.set_ylabel("Y")
#anim.save('Rosette.gif')
plt.show()
```

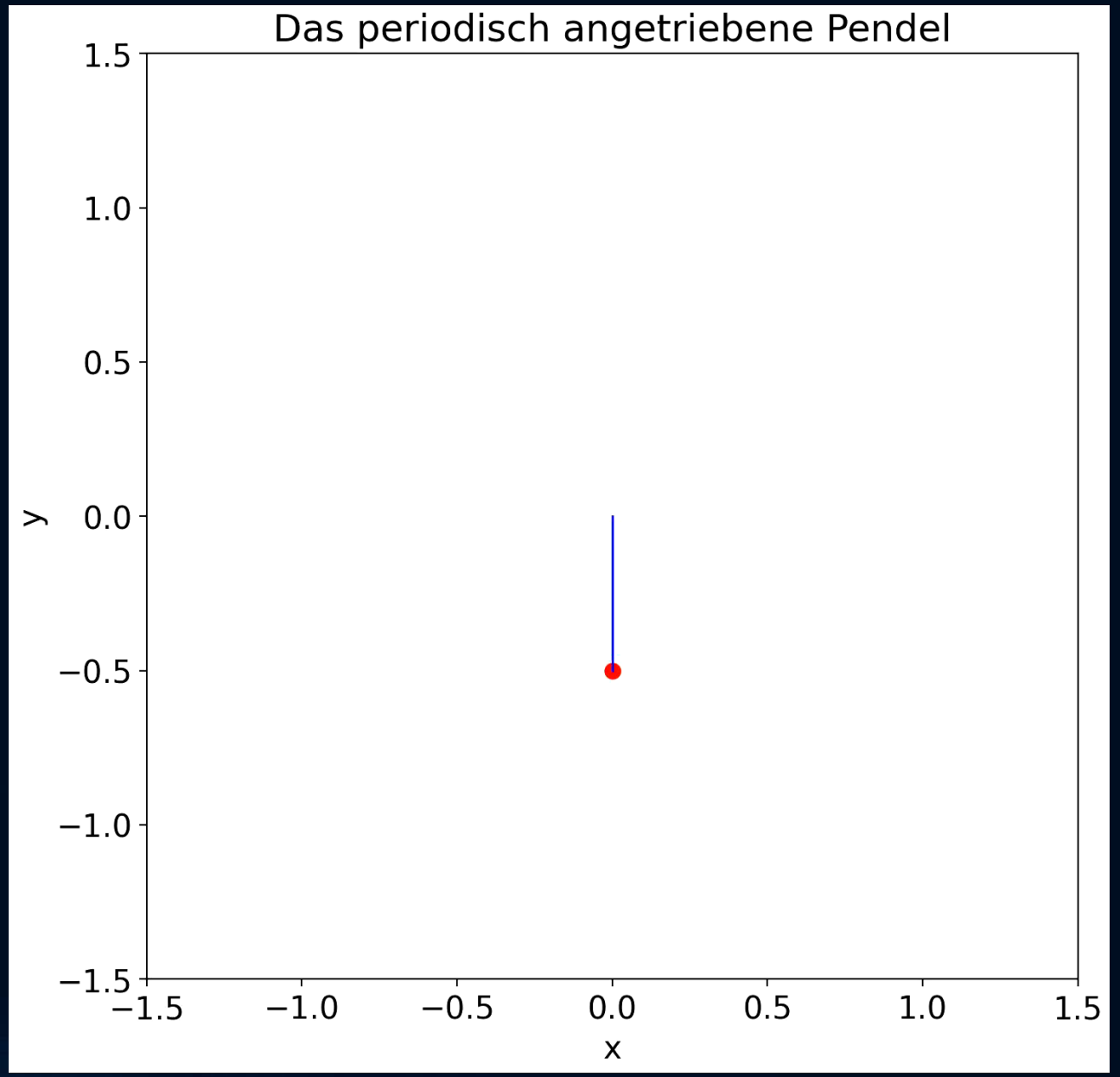
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL JUPYTER

```
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452

[Done] exited with code=0 in 23.518 seconds
```

Studentische Projekte

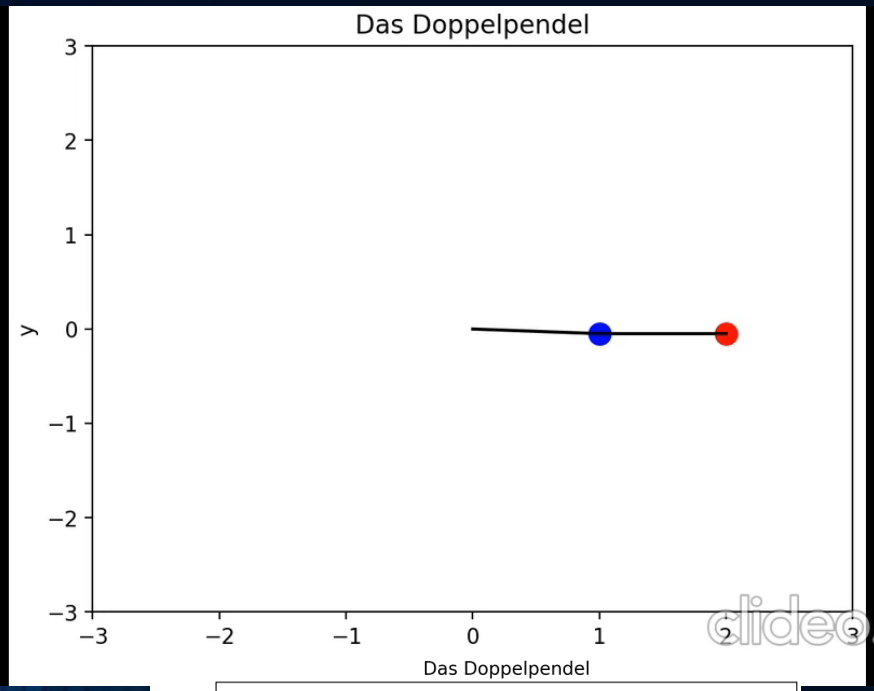
Das periodisch angetriebene Pendel

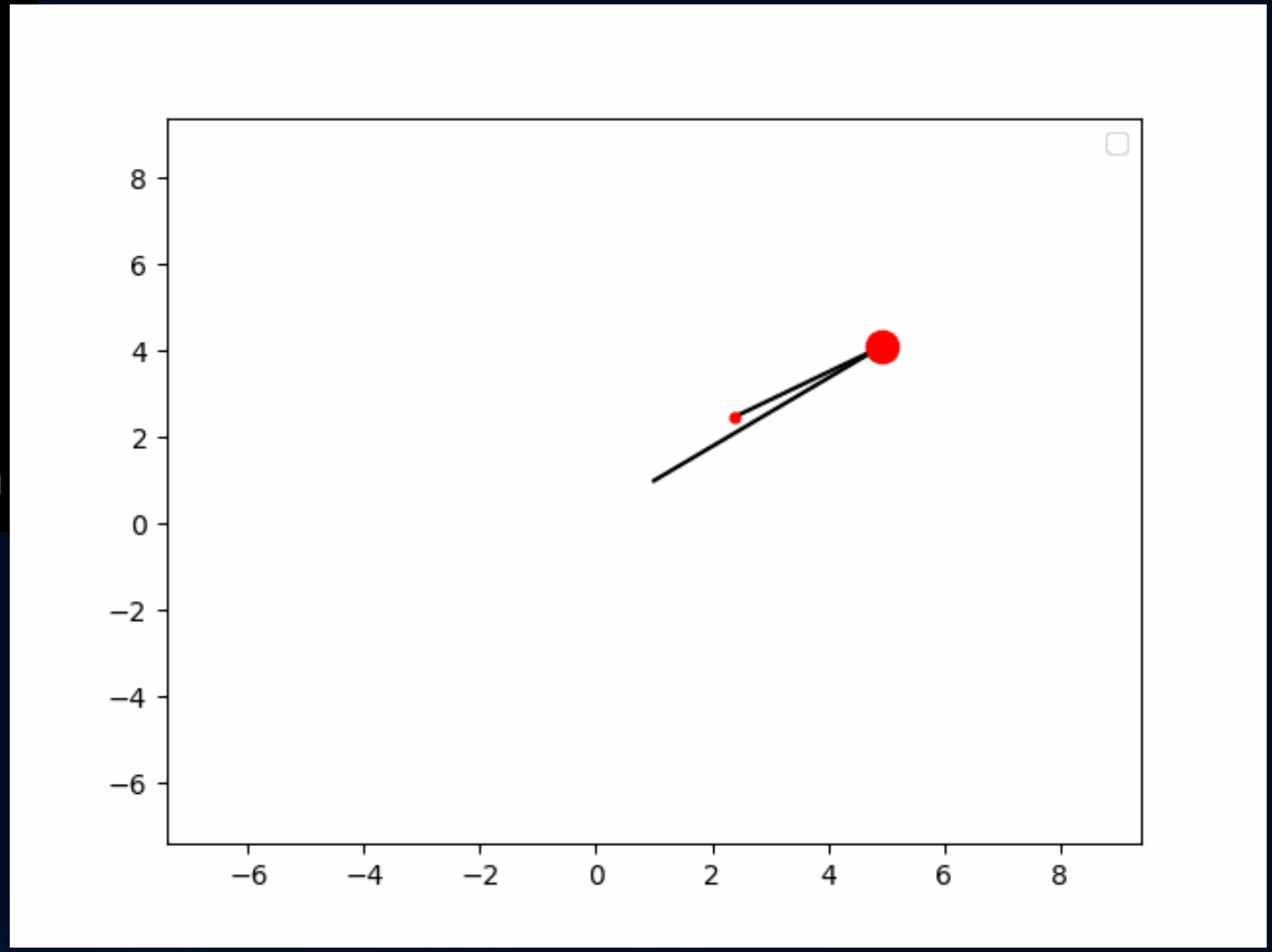


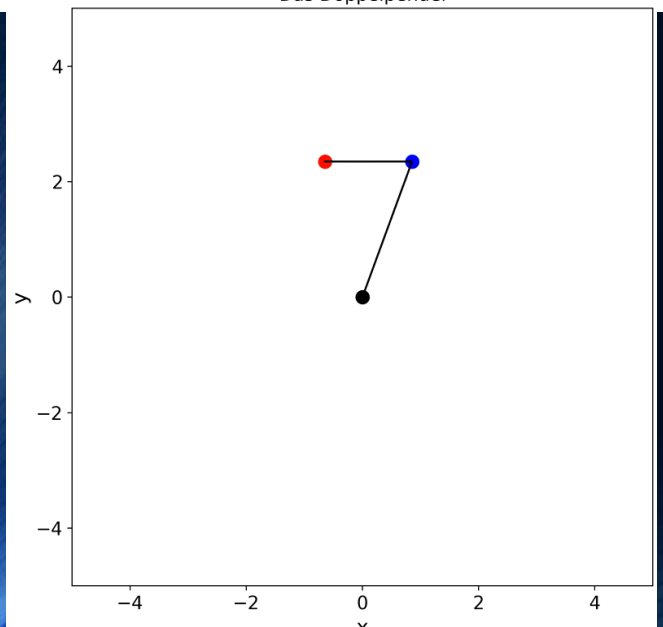


Studentische Projekte

Das Doppelpendel

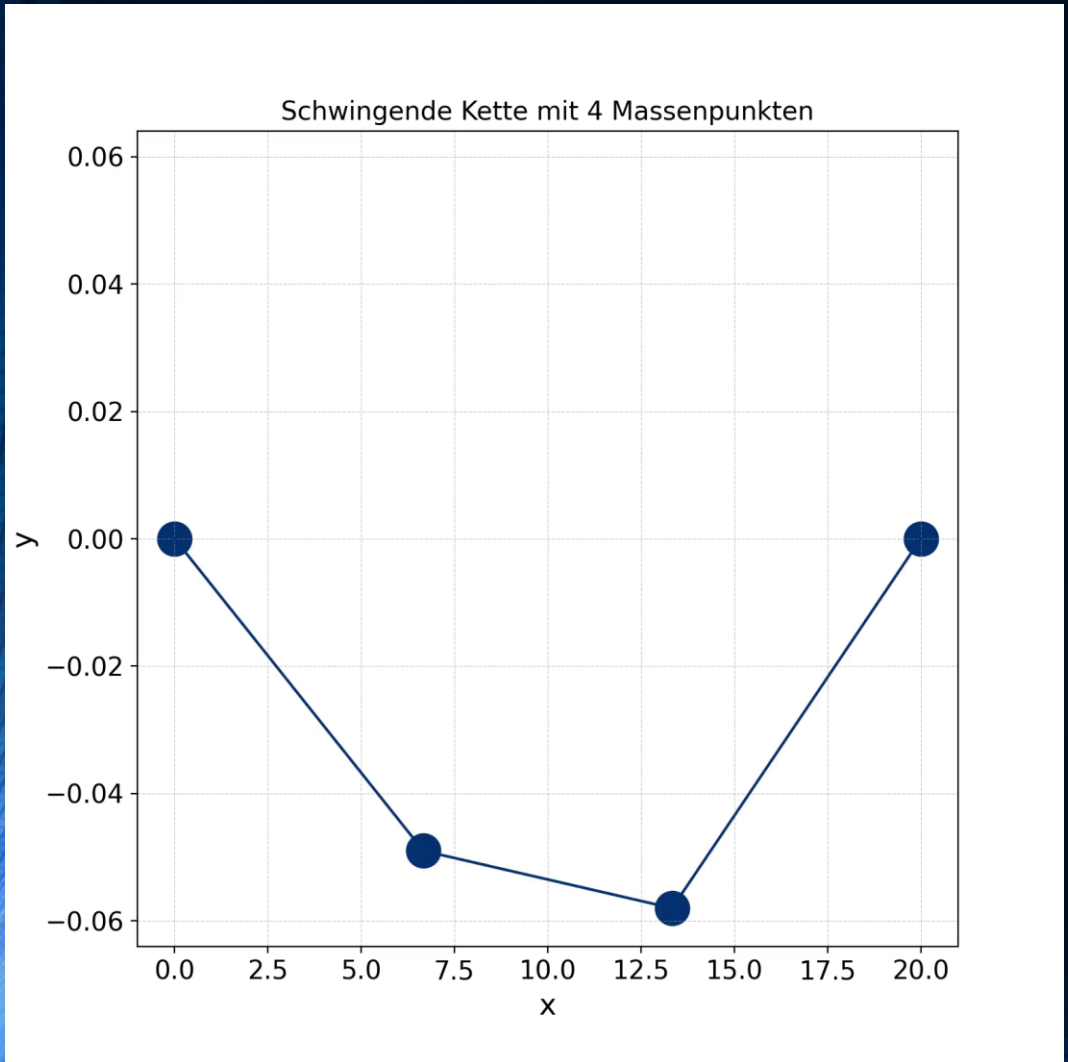


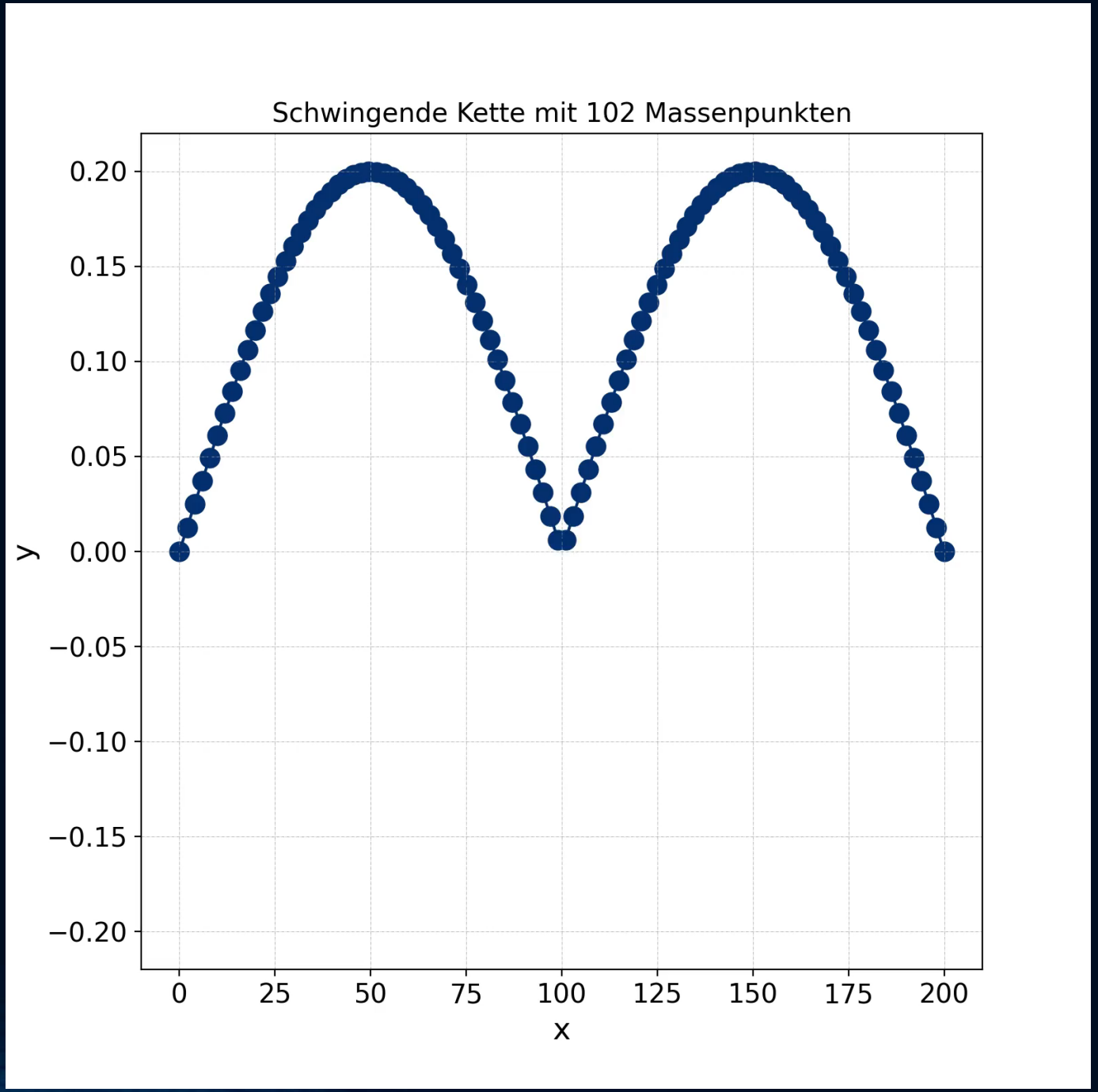




Studentische Projekte

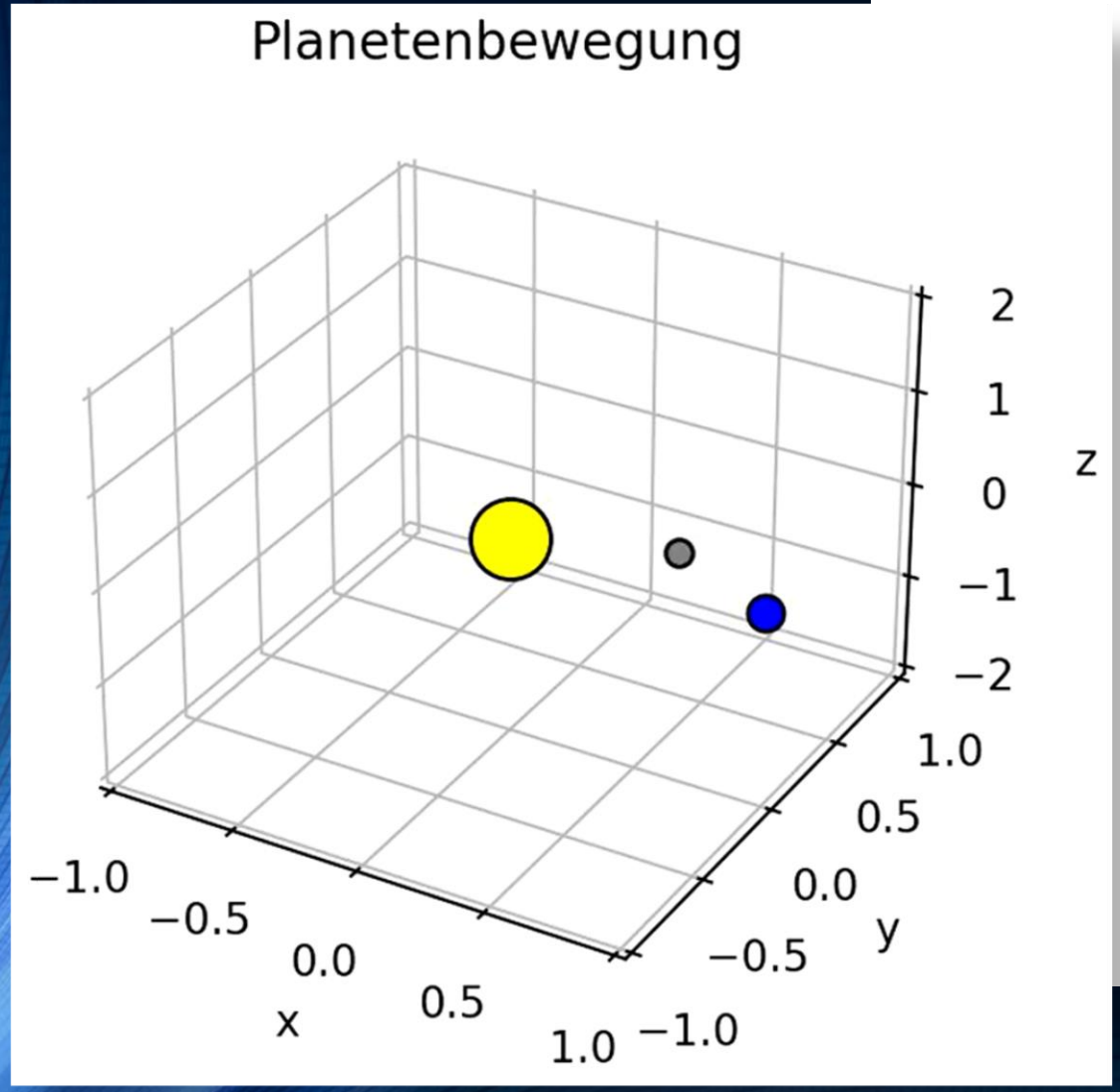
Die schwingende Kette

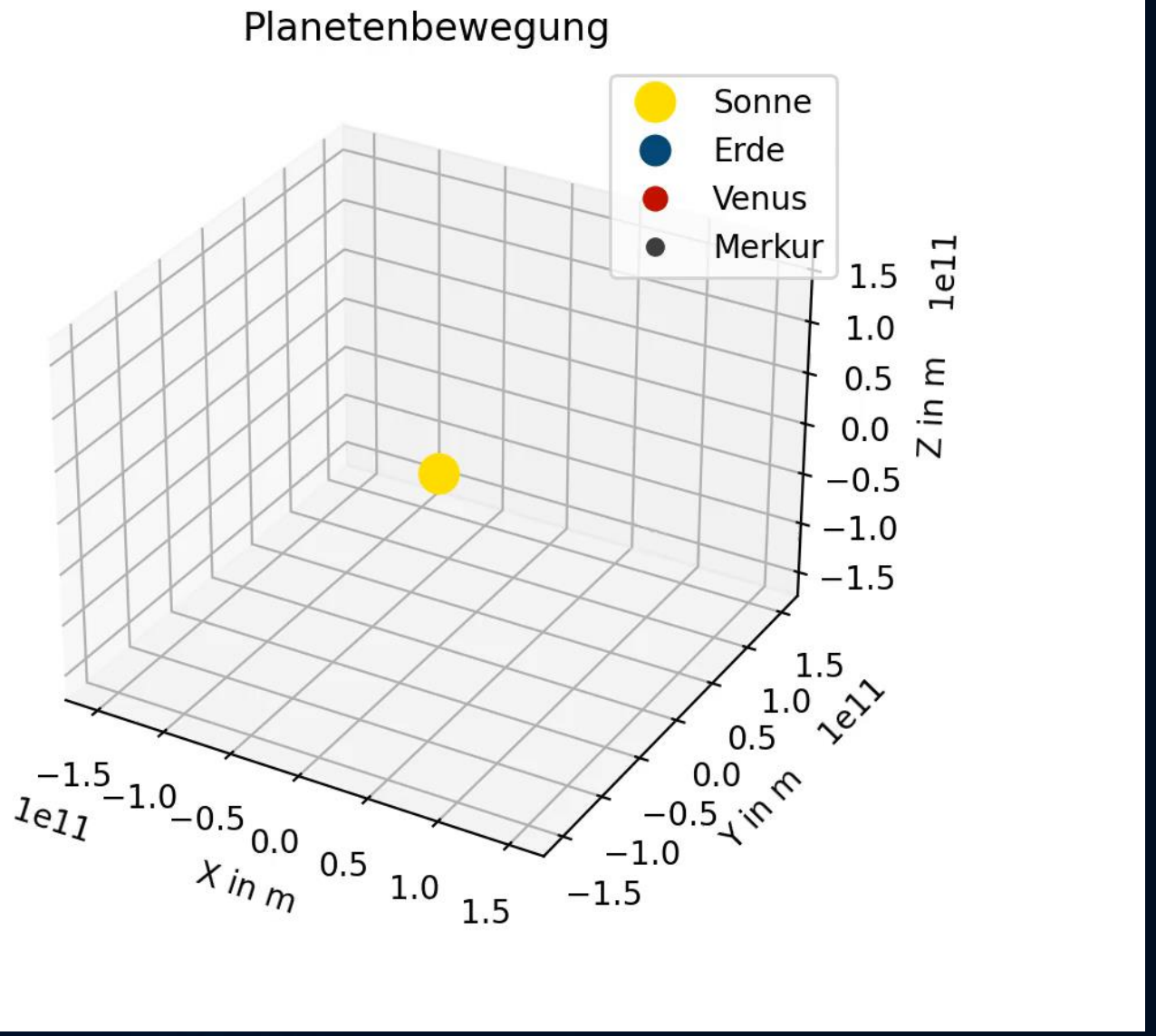




Studentische Projekte

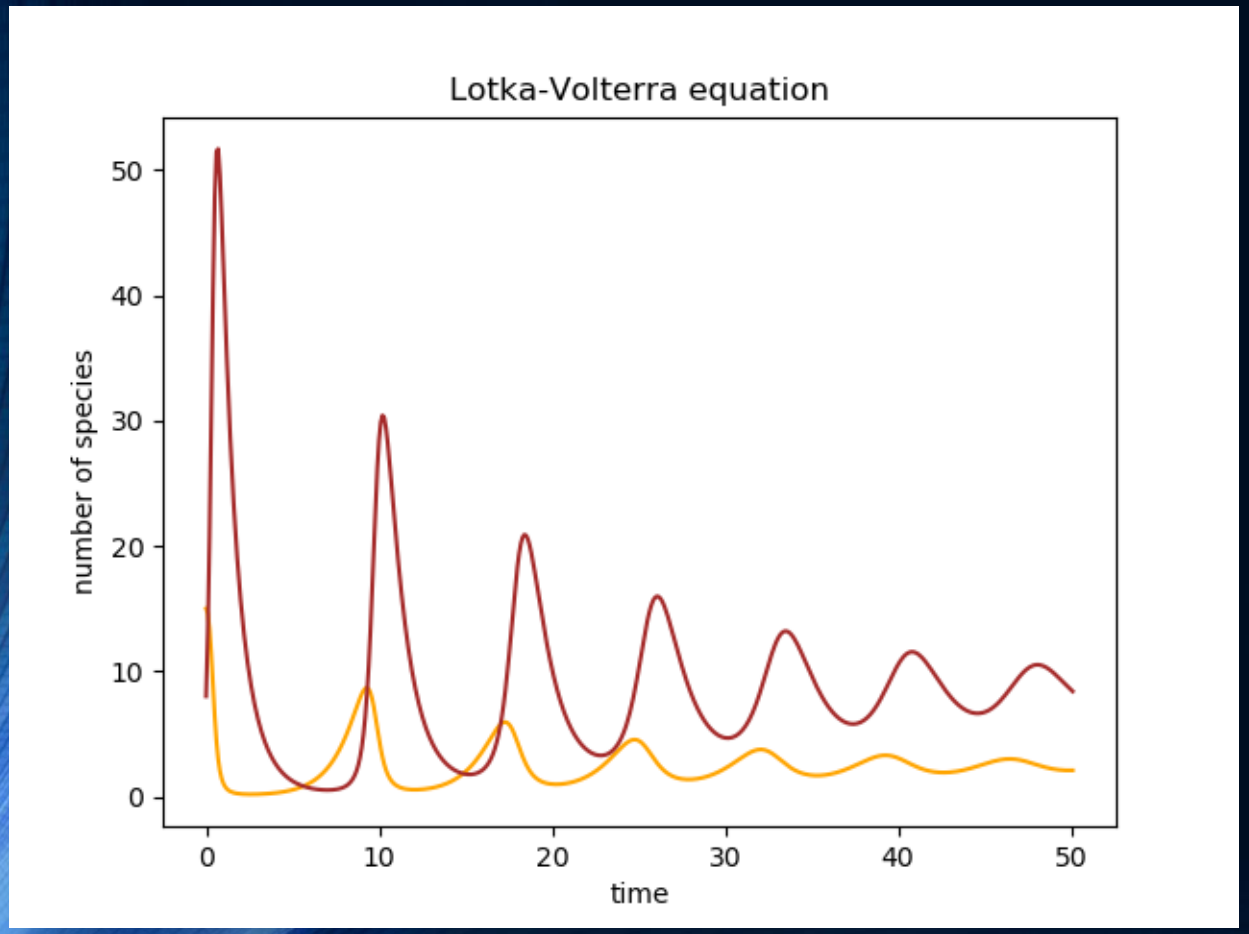
Planetenbewegungen

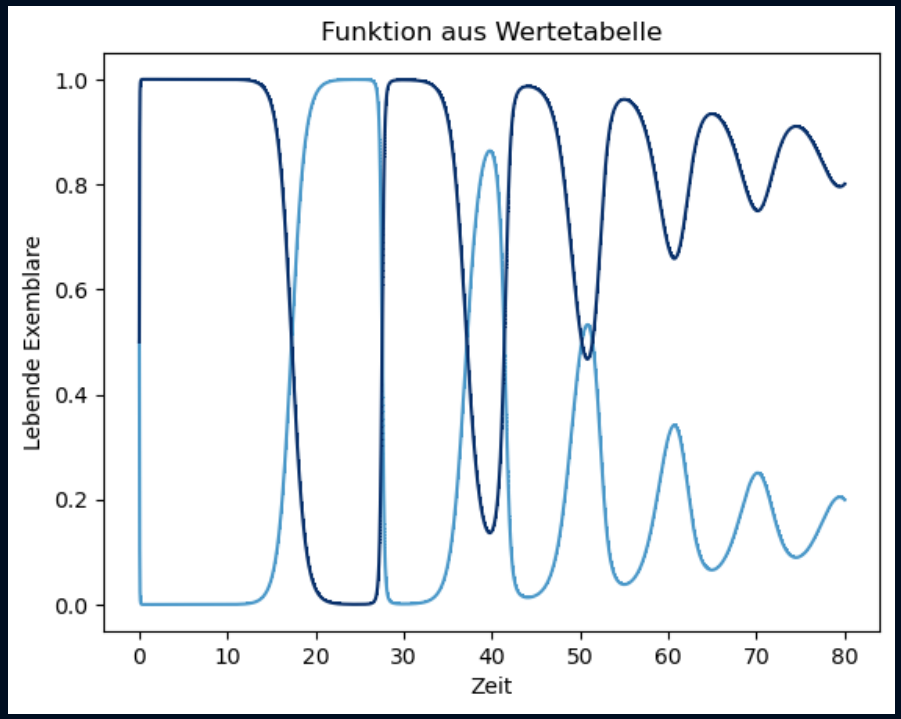


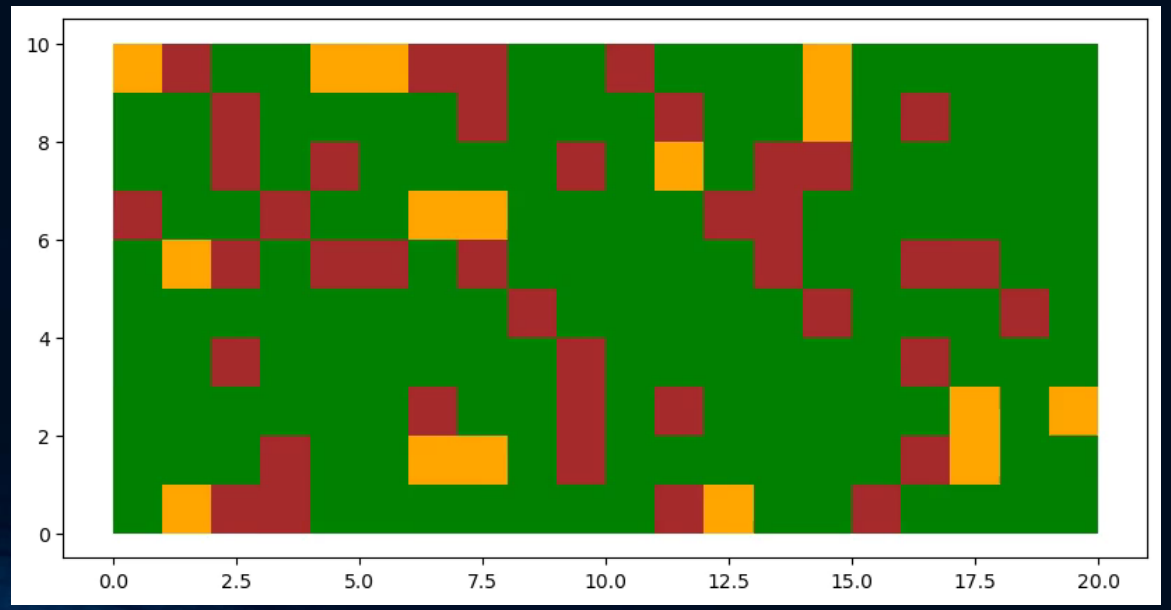


Studentische Projekte

Räuber-Beute Simulationen



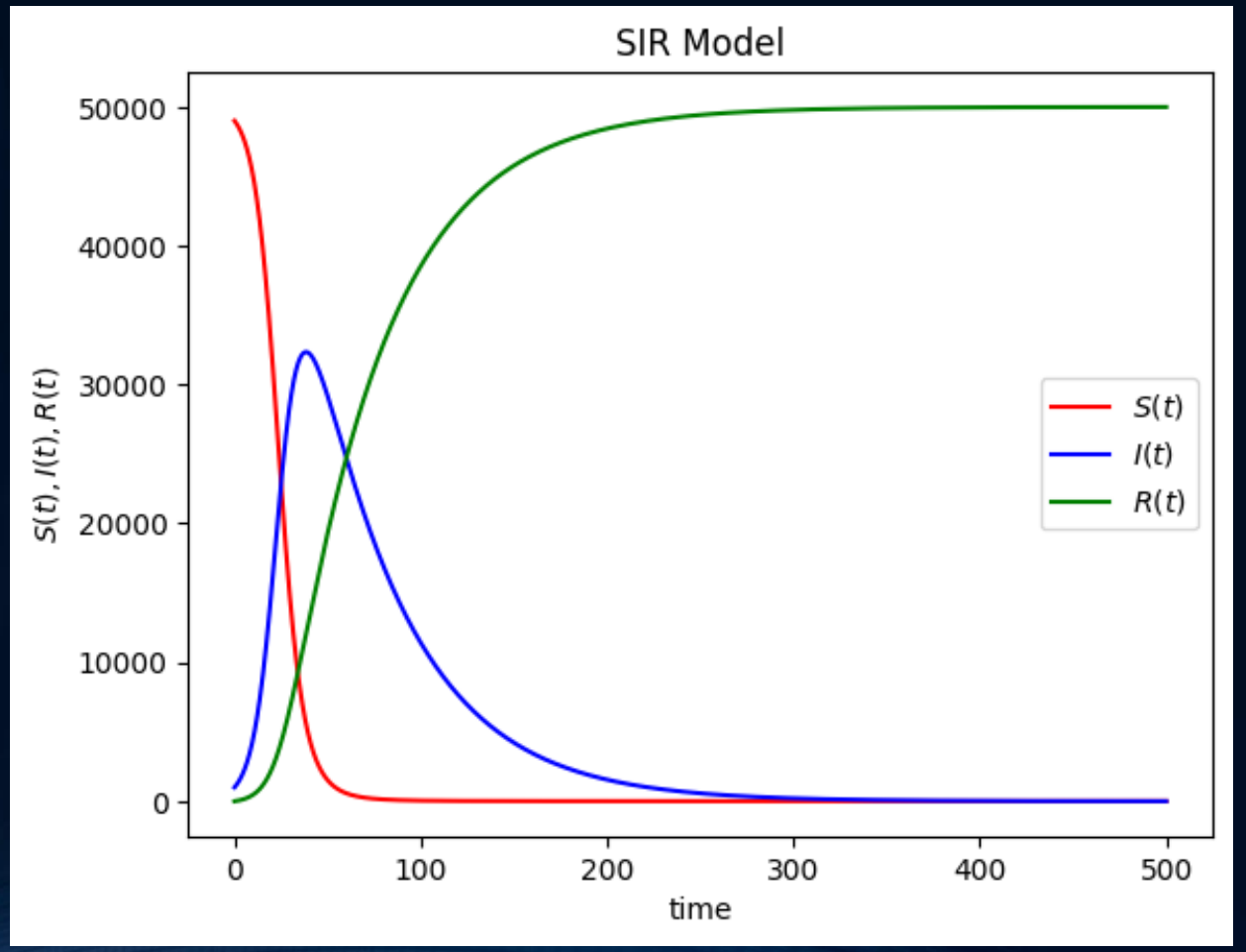




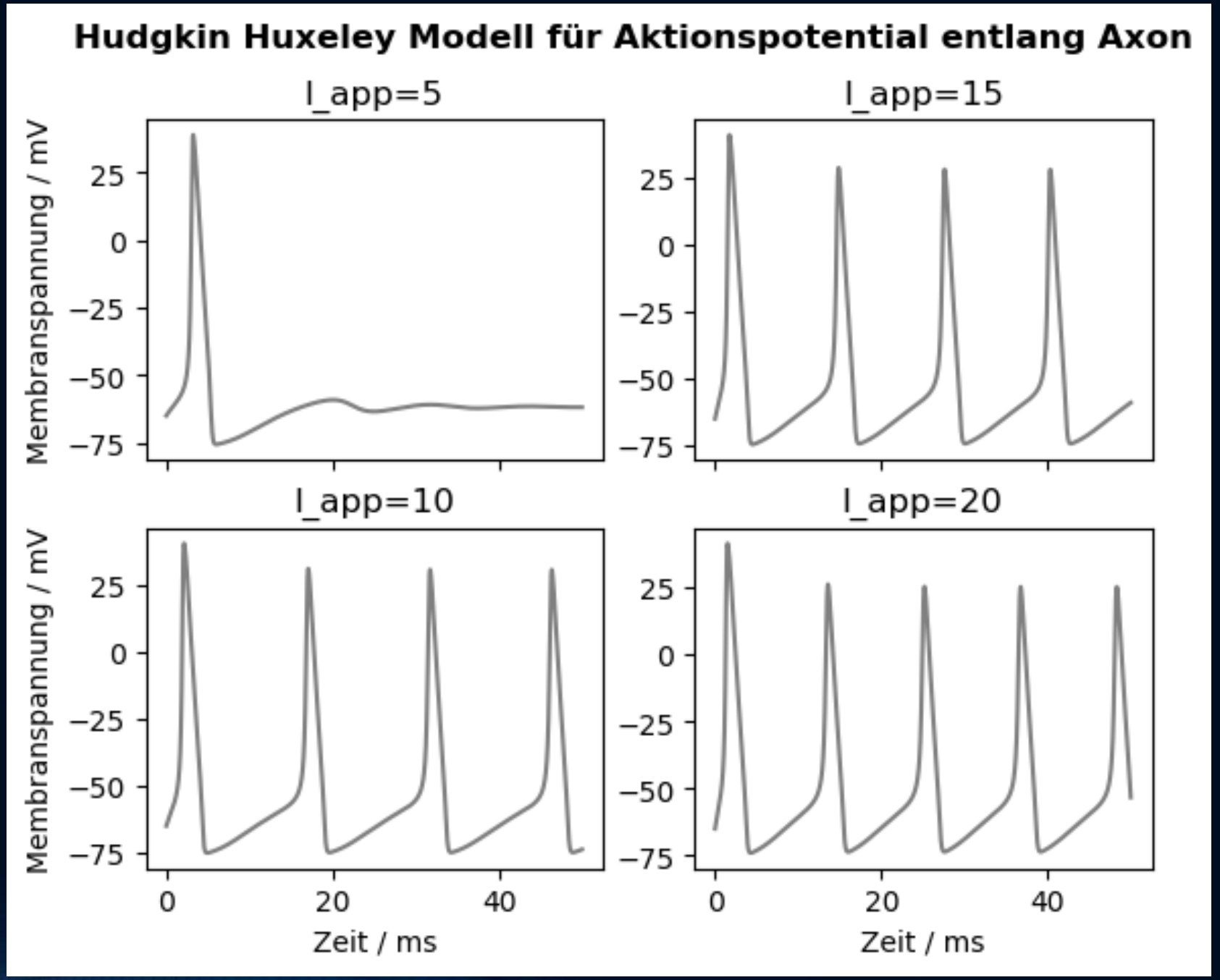
Studentische Projekte

Das Susceptible-Infected-Recovered (SIR) Modell

Ausbreitung eines Krankheitserregers (z.B. COVID-19 Vrius) auf einem komplexen Kontakt-Netzwerk

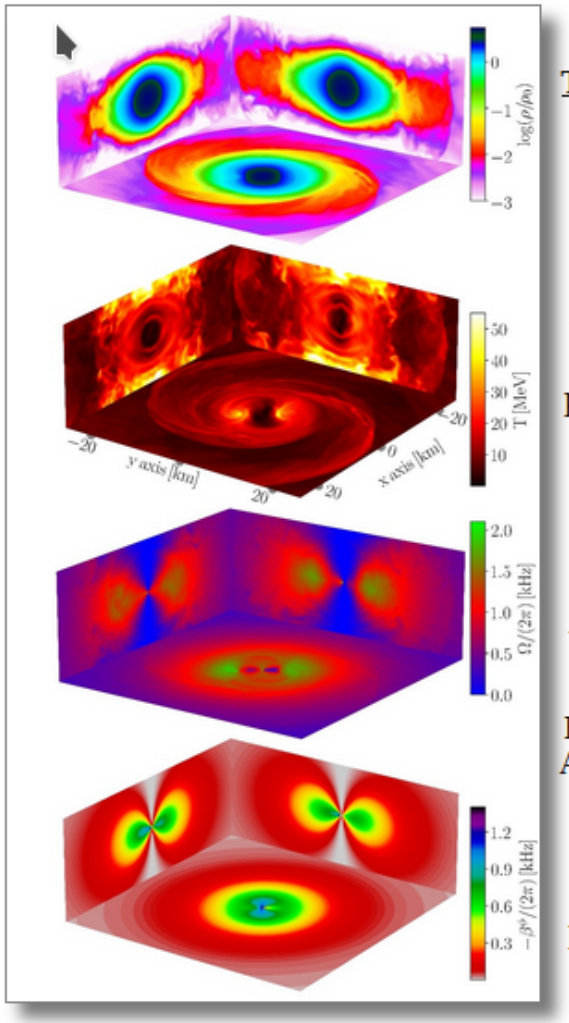


Studentische Projekte



Anwendungsbeispiele und weiterführende Vorlesungen

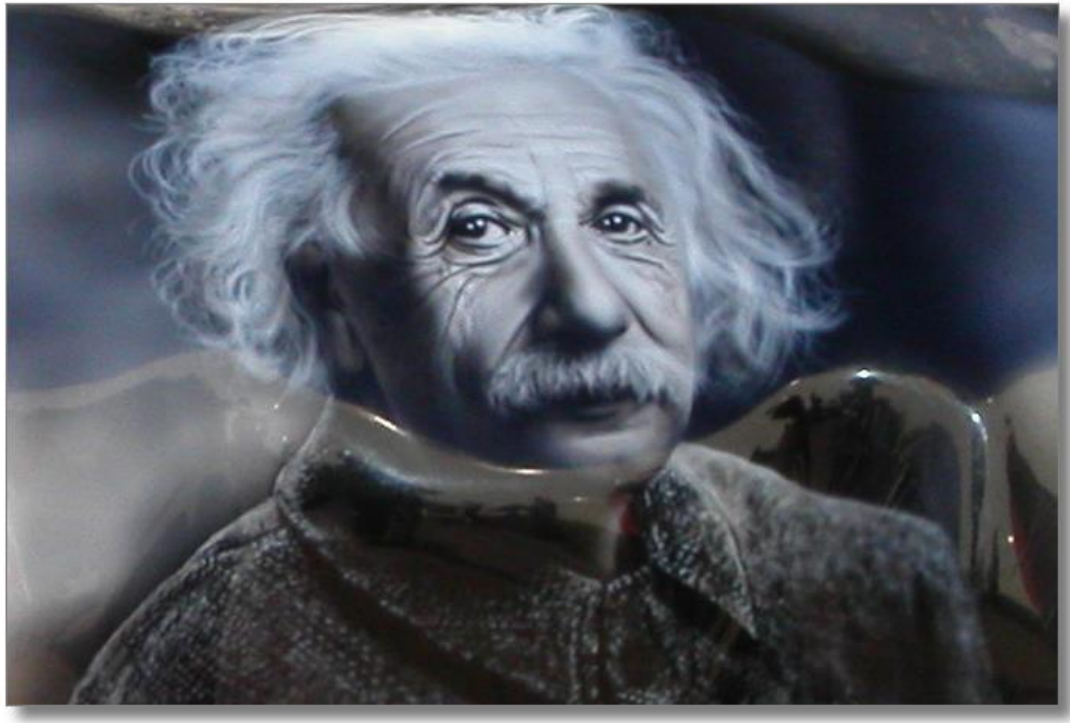
Die Anwendungsmöglichkeiten der Programmierung sind vielschichtig und speziell im Bereich der theoretischen Physik, sind viele der aktuellen Forschungsfragen nur mittels aufwendiger Computersimulationen lösbar.



Eines dieser aktuellen Anwendungsfelder ist die numerische *relativistische Astrophysik* und die *numerische allgemeine Relativitätstheorie*. Das Einstein Toolkit (ET) ist eine Softwareplattform, mit der man Probleme aus dem Bereich der relativistischen Astrophysik und Gravitationsphysik nummerisch am Computer simulieren kann. Das ET entwickelte sich im Jahre 1998 aus dem Cactus Code und im Laufe der letzten 20 Jahre wurde seine Performance (hocheffiziente OpenMP/MPI Parallelisierung) und Anwendungsbreite ständig verbessert. Mittlerweile kann man mit dem frei zugänglichen ET-Code Kollisionen von schwarzen Löchern und Neutronensternen simulieren. Die nebenstehende Abbildungund zeigt einige Eigenschaften (Dichte, Temperatur, Rotationsprofil und Frame-Dragging) des in einer Neutronenstern-Kollision produzierten hypermassiven Neutronensterns und wurde mit dem ET simuliert und mittels Python visualisiert. Näheres siehe Vorlesung 'Allgemeine Relativitätstheorie mit dem Computer' und F+L-Praktikum am Institut für Theoretische Physik.

Der wissenschaftliche Untersuchungsgegenstand des zweiten aktuellen Anwendungsfeldes der Programmierung ist, im Gegensatz zum ersten, ein der menschlichen Wahrnehmung näheres. Die Spieltheorie ist ein bedeutendes Mittel zum modellhaften Verständnis komplexer sozioökonomischer Entscheidungs-Prozesse. In ihrer herkömmlichen Form werden die handelnden Akteure als rein rational und egoistisch handelnde Individuen aufgefasst (homo oeconomicus) und eine Population bestehend aus einer großen Anzahl von solchen interdependenten Entscheidern kann sich zu einem Dilemma-artigen Zustand entwickeln. Im Gegensatz zur Physik, die die Gesetzmäßigkeiten der leblosen Materie/Energie und deren Wechselwirkungen in Raum und Zeit betrachtet, ist in der *evolutionären Spieltheorie* das zu erforschende Ding, der Mensch, und wir sind daran interessiert, wie dieser sich bei ökonomischen oder sozial relevanten strategischen Entscheidungen verhält. Viele Wirtschafts- und Sozialwissenschaftler betrachten die *Spieltheorie* als die formale Sprache der ökonomischen Theorie. Agentenbasierte Computersimulationen von spieltheoretischen Problemen auf komplexen sozio-ökonomischen Netzwerken sind ein interdisziplinäres aktuelles Forschungsfeld; näheres siehe Vorlesung Physik der sozio-ökonomischen Systeme mit dem Computer im kommenden Wintersemester 2022/23.

Vorwort



Die Vorlesung *Allgemeine Relativitätstheorie mit dem Computer* wurde im Sommersemester 2016 das erste Mal gehalten und viele der auf dieser Hauptseite erreichbaren Internetseiten basieren grundsätzlich auf dem damals erstellten Kurs. In der Vorlesung werden die mathematisch anspruchsvollen Gleichungen der Allgemeinen Relativitätstheorie (ART) in diversen Programmierumgebungen analysiert. Im ersten Teil des Kurses erlernen die Studierenden die Verwendung von Computeralgebra-Systemen (Python Jupyter Notebooks, Maple und Mathematica). Die oft komplizierten und zeitaufwendigen Berechnungen der tensoriellen Gleichungen der ART können mithilfe dieser Programme erleichtert werden. Diverse Anwendungen der Einstein- und Geodätengleichung werden in Jupyter Notebooks (eine Open-Source, web-basierte interaktive Programmierumgebung) und Maple Worksheets implementiert, quasi analytische Berechnungen durchgeführt und entsprechende Lösungen berechnet und visualisiert. Der zweite Teil des Kurses befasst sich mit der numerischen Berechnung von Neutronensternen und Weißen Zwergen mittels eines C/C++ Programms. Nach einer kurzen Auffrischung der grundlegenden Programmierkenntnisse, wird ein Programm besprochen, das die Tolman-Oppenheimer-Volkoff Gleichung numerisch löst und die Ergebnisse werden visualisiert. Zusätzlich wird hierbei in die Grundkonzepte der parallelen Programmierung eingeführt und eine MPI- und OpenMP-Version des C/C++ Programms erstellt. Im dritten Teil des Kurses werden zeitabhängige numerische Simulationen der ART mittels des Einstein Toolkit durchgeführt und deren Ergebnisse mittels Python/Matplotlib visualisiert. Inhaltlich wird hierbei ebenfalls auf den, dem Programm zugrunde

Allgemeine Relativitätstheorie mit dem Computer (General Theory of Relativity on the Computer) Vorlesung SS 2021

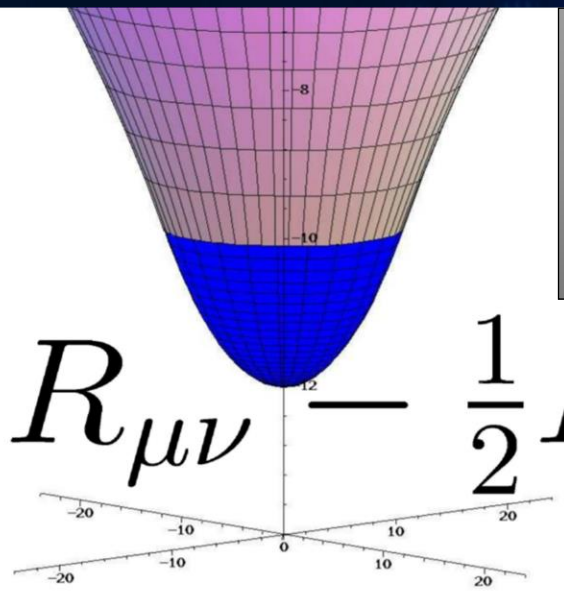
Aufgrund der andauernden Corona-Krise findet die Vorlesung und die Übungstermine auch in diesem Semester nur Online statt!

Diese Internetseite fasst die Online-Angebote der Vorlesung *Allgemeine Relativitätstheorie mit dem Computer* zusammen. Auf der linken Seite finden Sie die einzelnen Vorlesungsaufzeichnungen (Videos), Vorlesungspräsentationen (pdf-Dateien) und weiterführende Links. Die Vorlesungstermine (Zoom Meetings, synchrones Lehrangebot) finden jeweils freitags von 15.00-17.00 Uhr statt. An den Online-Übungen können Sie entweder freitags vor (13.30-15.00 Uhr) oder nach der Vorlesung (17:00-18:30 Uhr) teilnehmen. Alle Lehrangebote werden mittel der Zoom Meeting Software gemacht und die jeweiligen Zoom-Links sind in der rechten oberen Ecke dieser Internetseite angegeben.
Die Inhalte der Vorlesung gliedern sich in drei Teile (Teil I, Teil II, Teil III), die Sie in der zweiten oberen Spalte einsehen können. Weiteres Zusatzmaterial und diverse Online-Aufgaben sind über die Online-Lernplattformen OLAT und Lon Capa erhältlich (siehe *E-Learning*).
Der Schwerpunkt der gesamten interaktiven Vorlesung liegt sowohl auf der Allgemeine Relativitätstheorie als auch auf der Vermittlung spezieller Programmierkenntnisse. Bei regelmäßiger Teilnahme an der Online-Vorlesung und den zugehörigen Übungseinheiten kann man einen benoteten bzw. unbenoteten Schein mit fünf Creditpoints zu erhalten.

Weiterführende Literatur

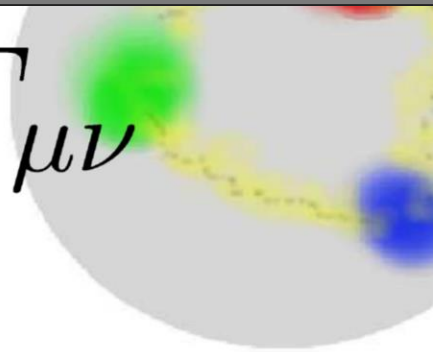
- *General relativity : An introduction for physicists* von M. P. Hobson, G. P. Efstathiou und

Grundlagen der Allgemeinen Relativitätstheorie



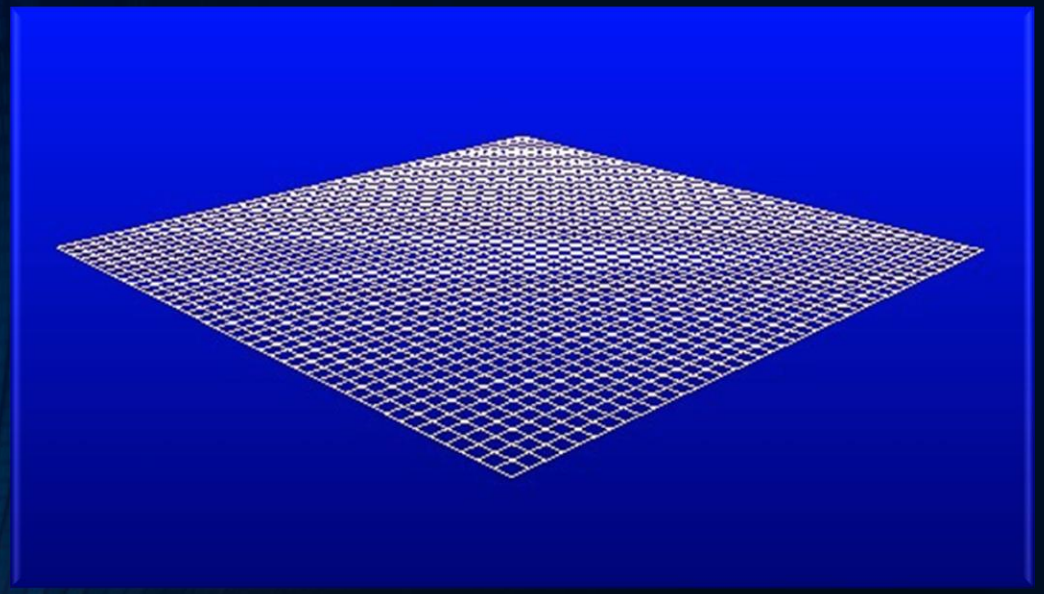
Vor etwa hundert Jahren (1915) stellte Albert Einstein seine „Allgemeine Relativitätstheorie“ (ART) der Öffentlichkeit vor.

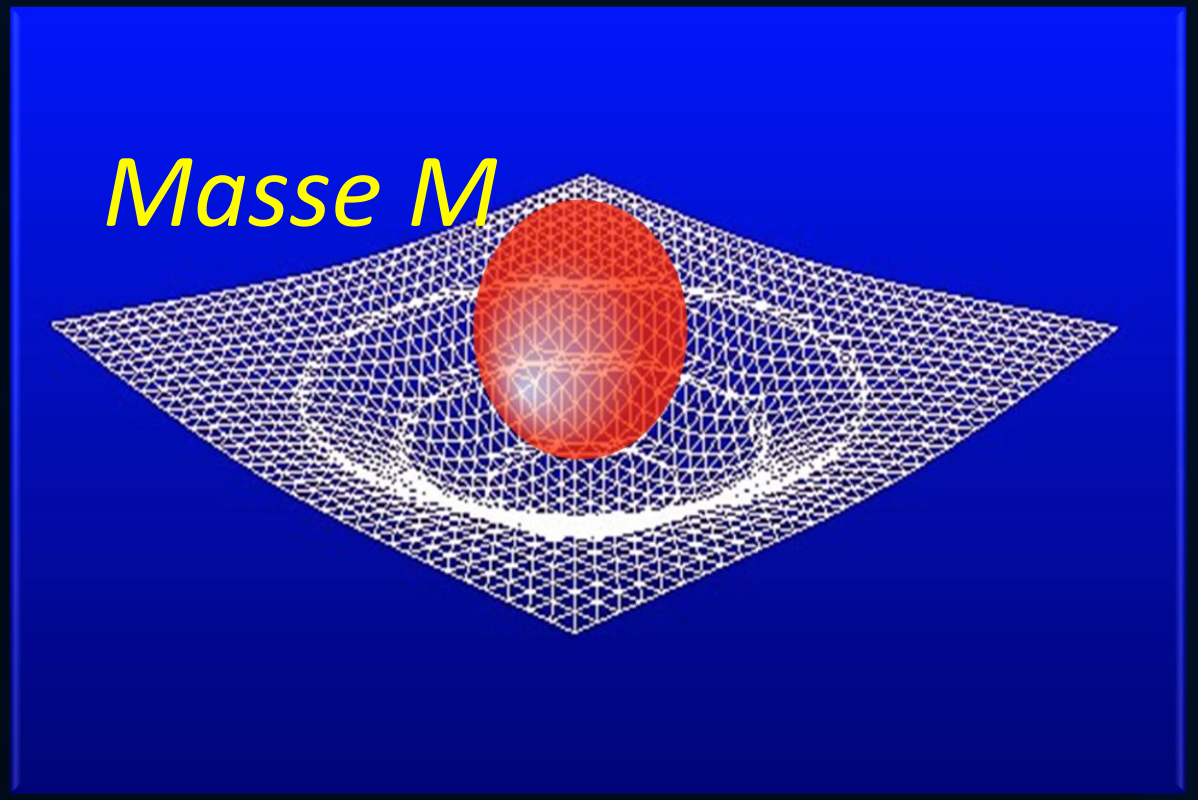
$$R_{\mu\nu} - \frac{1}{2} R\, g_{\mu\nu} = \frac{8\pi G}{c^4} T_{\mu\nu}$$



Die ART ist eine sehr revolutionäre Theorie. Sie besagt, dass jegliche Energieformen (z.B. Masse eines Körpers) die „Raumzeit“ verbiegen und durch diese Krümmung des Raumes und der Zeit die Gravitation (Schwerkraft) resultiert. -> Raumzeit-Krümmung = Energie

Was ist Raumzeit-Krümmung?





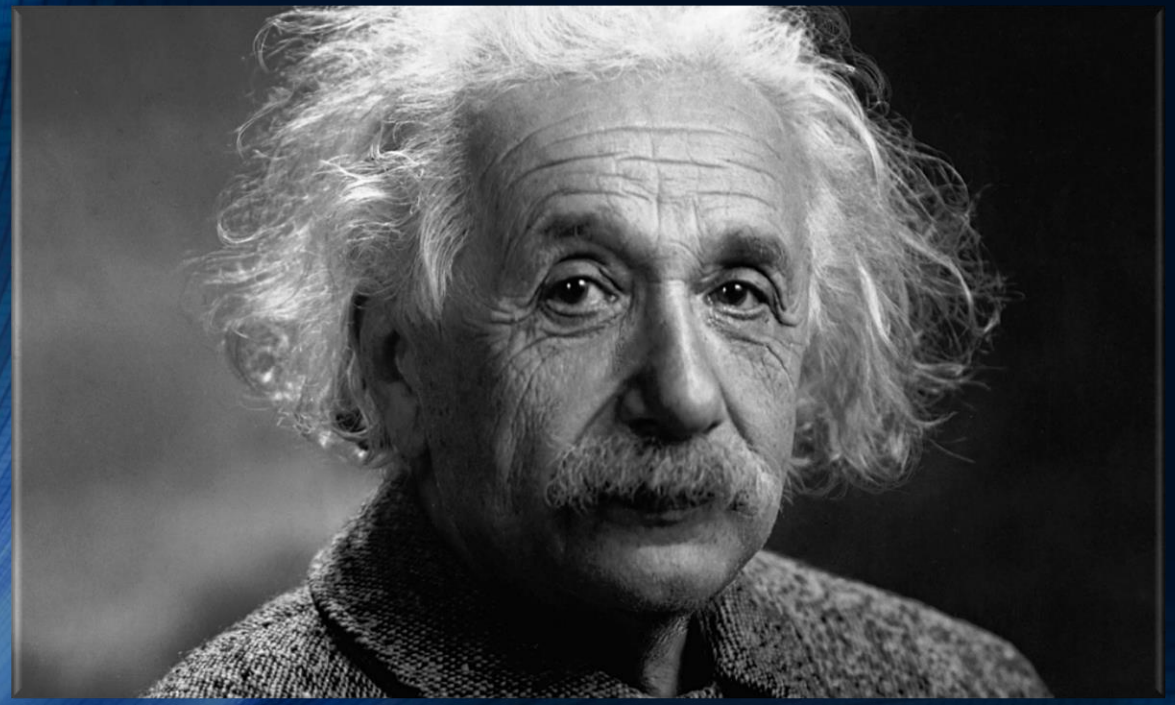
Flache Raumzeit
Raumzeit ohne Materie und Energie hat keine Krümmung

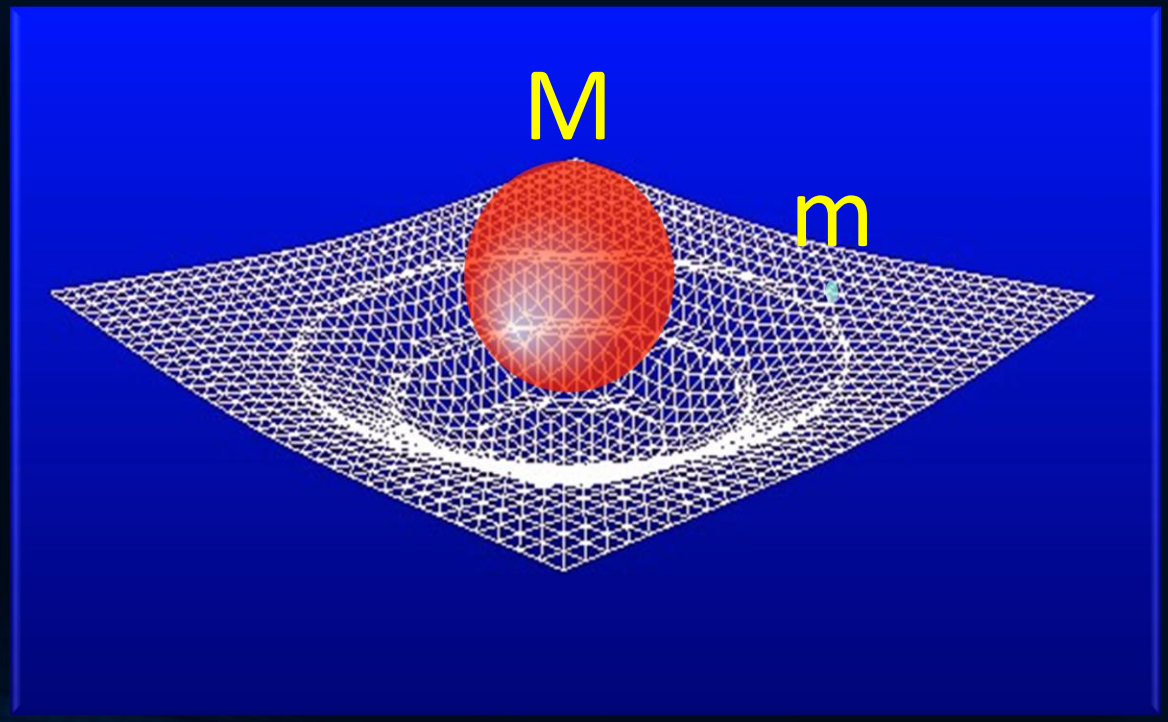
Gekrümmte Raumzeit
Raumzeit mit Materie verbiegt sich

Raumzeit-Krümmung ist Gravitation?

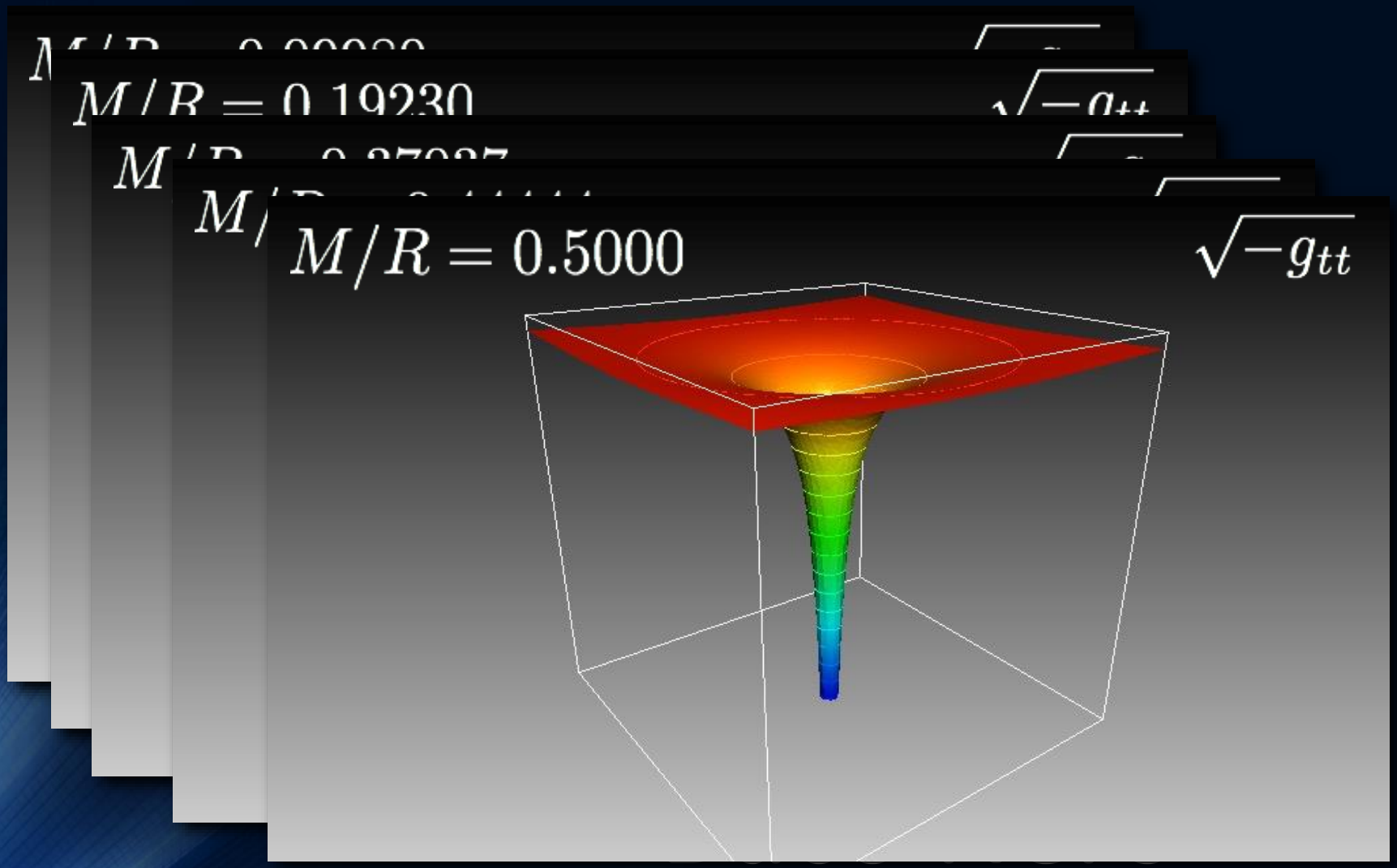
Betrachten wir einen Objekt kleiner Masse m das um ein Objekt großer Masse M kreist (z.B. Erde um die Sonne)

Einstein: Die Krümmung der Raumzeit, verursacht durch die große Masse, bestimmt die Umlaufbahn des kleinen Körpers und ist ursächlicher Grund der gravitativen Wechselwirkung





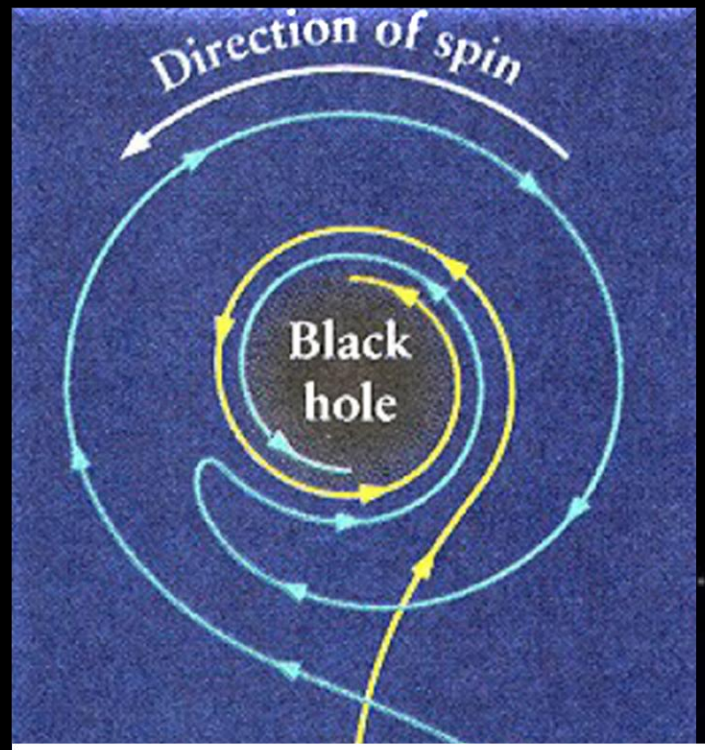
Was sind schwarze Löcher?

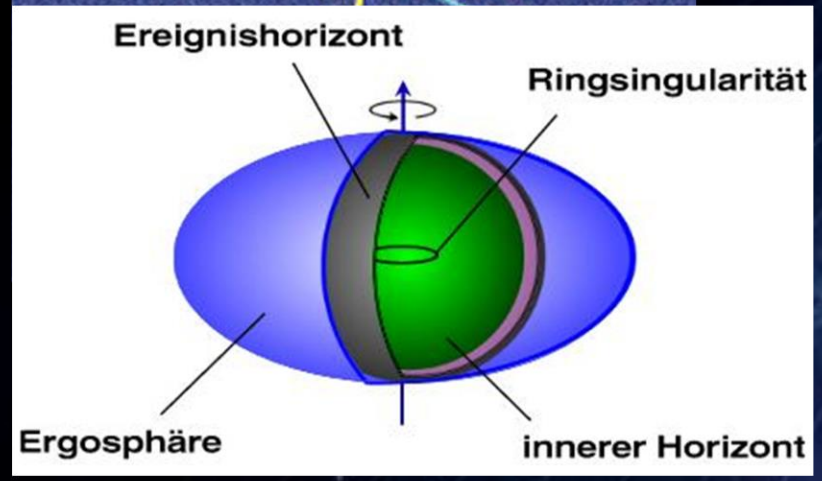


Wir sind über den Grenzwert gekommen und haben ein schwarzes Loch erzeugt!

Grenzwert der Krümmung: Stabile Objekte (Neutronensterne) sind nicht mehr möglich

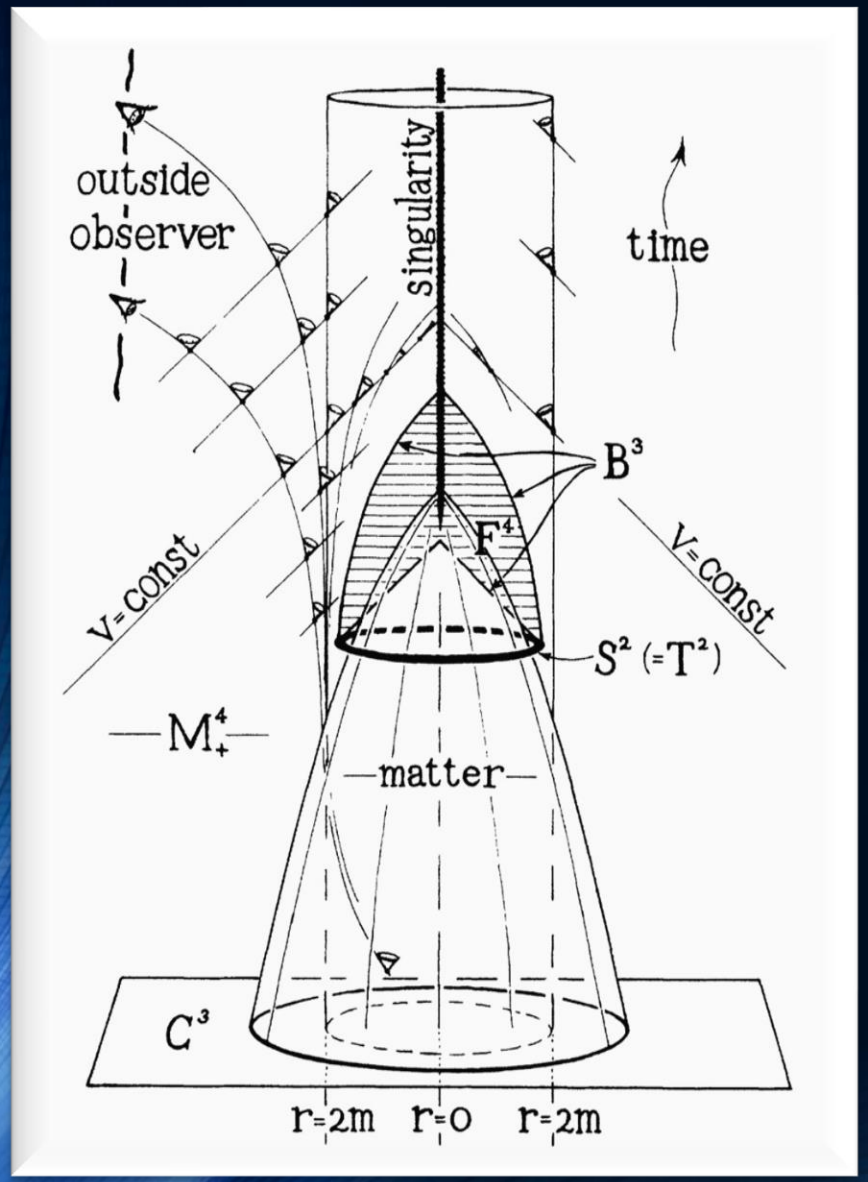
Rotierende schwarze Löcher



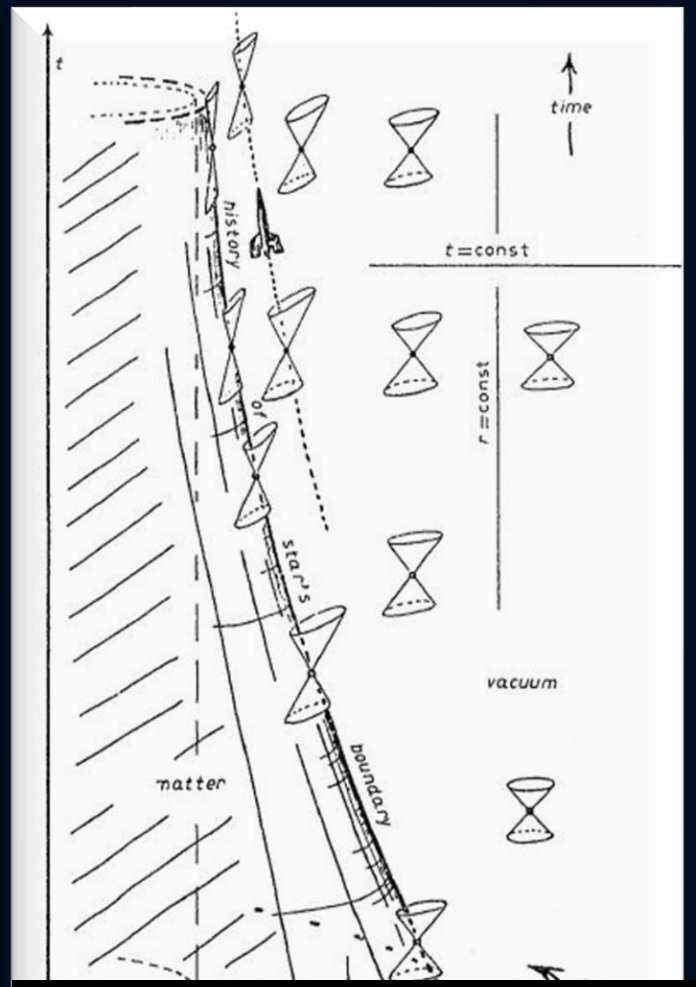


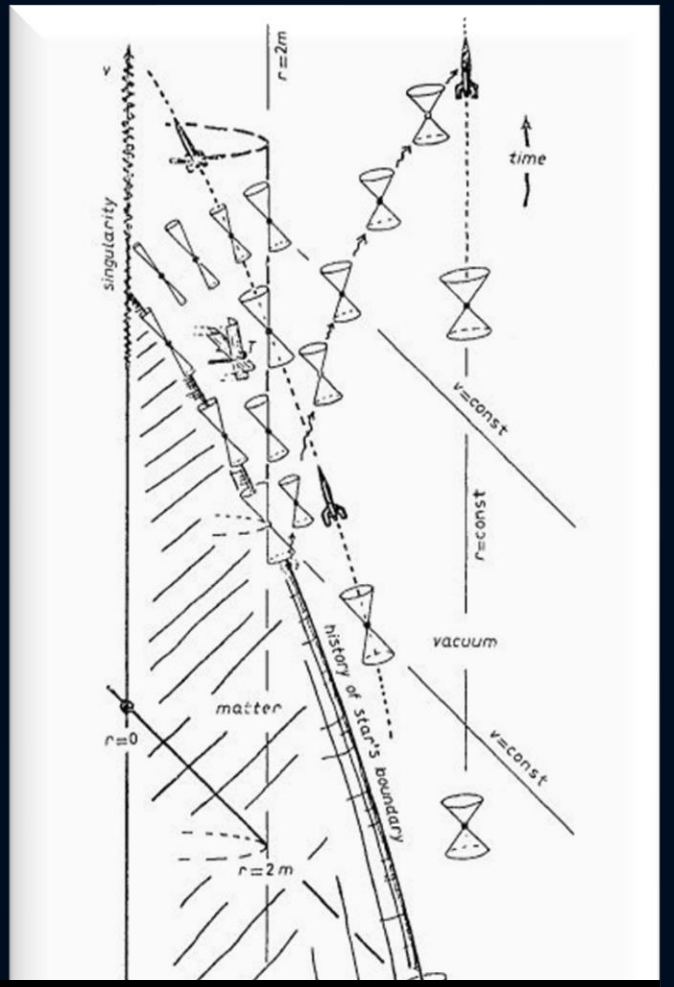
Wie entstehen schwarze Löcher ?

GRAVITATIONAL COLLAPSE AND SPACE- TIME SINGULARITIES
Nobel Price 2020: R.Penrose, PRL Vol.14 No.3 (1965)



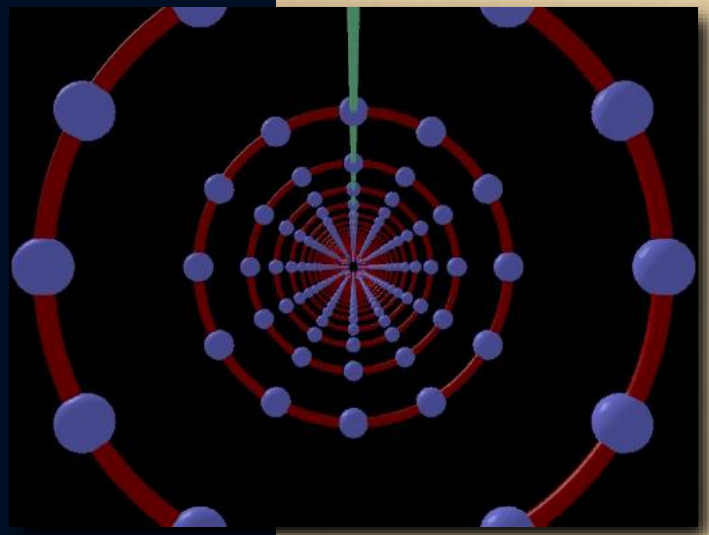
Self-drawn space-time diagram by R.Penrose (1965)

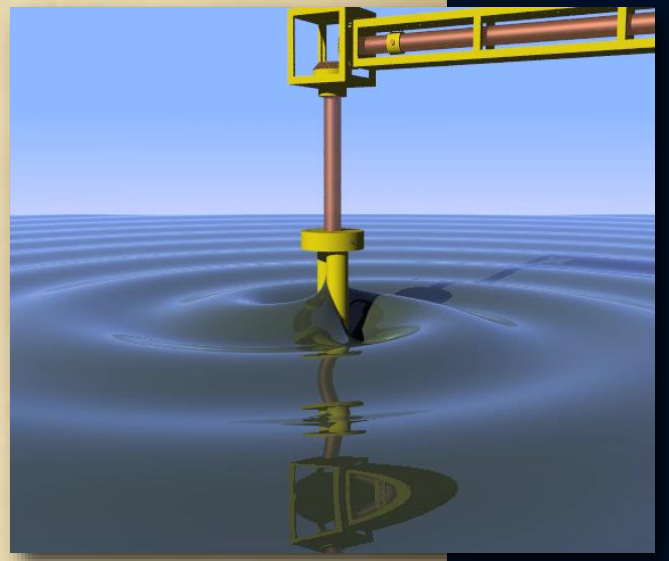




R. Penrose: Nobel Preis 2020 für die Entdeckung, dass die Bildung von Schwarzen Löchern eine robuste Vorhersage der allgemeinen Relativitätstheorie ist

R.Penrose in Rivista del Nuovo Cimento, Num.Spez. I, 257 (1969)





Über Gravitationswellen.

Von A. Einstein.

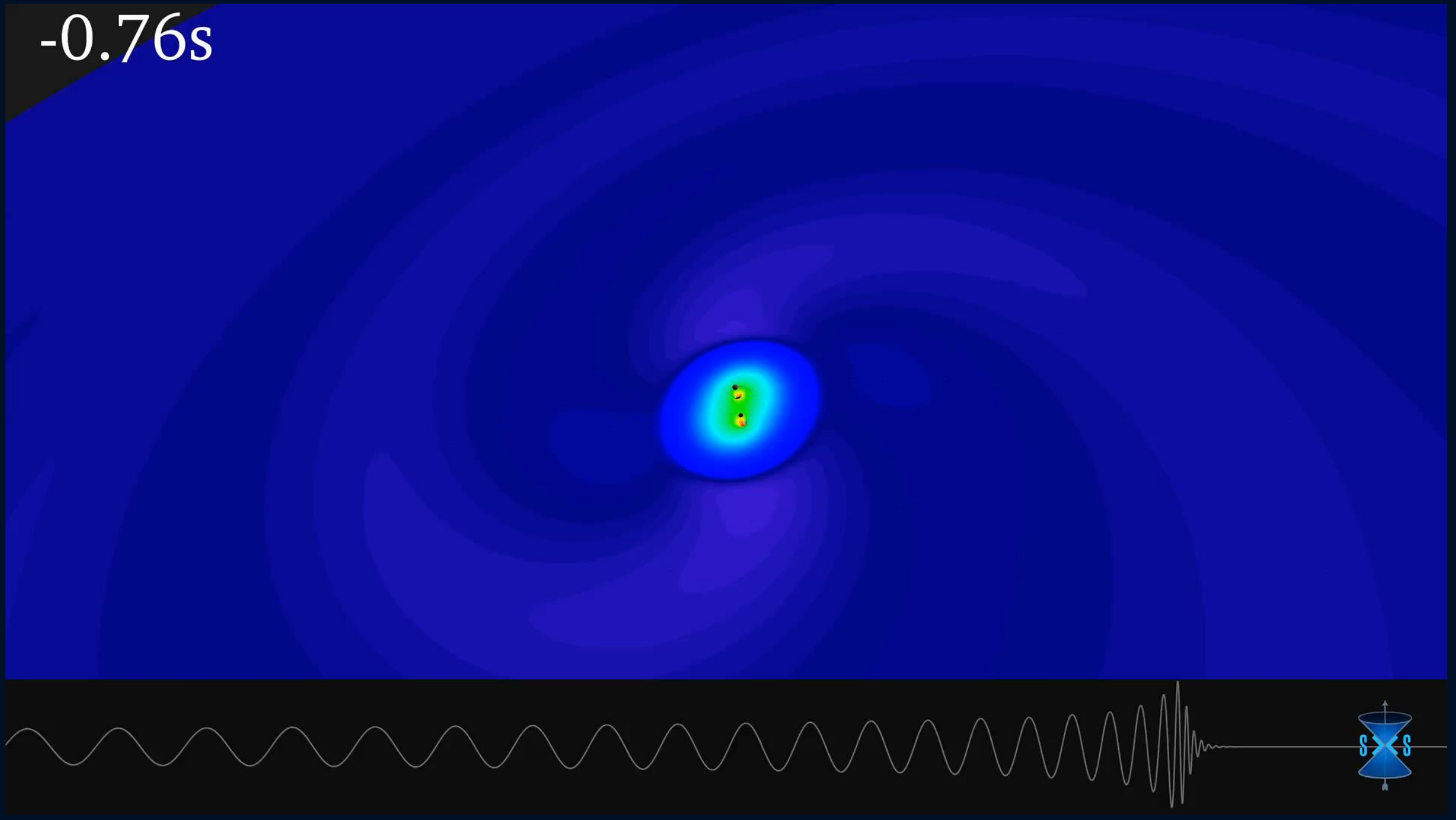
(Vorgelegt am 31. Januar 1918 [s. oben S. 79].)

Die wichtige Frage, wie die Ausbreitung der Gravitationsfelder erfolgt, ist schon vor anderthalb Jahren in einer Akademiearbeit von mir behandelt worden[1]. Da aber meine damalige Darstellung des Gegenstandes nicht genügend durchsichtig und außerdem durch einen bedauerlichen Rechenfehler verunstaltet ist, muß ich hier nochmals auf die Angelegenheit zurückkommen.

Einsteins erste Arbeit über Gravitationswellen

Sitzungsberichte der Königlich-Preußischen Akademie der Wissenschaften

Kollidierende Schwarze Löcher



100 Jahre später LIGO:
LIGO: **Laser Interferometer Gravitational-Wave Observatory**

PRL **116,** 061102 (2016)

Selected for a Viewpoint in *Physics*
PHYSICAL REVIEW LETTERS

week ending
12 FEBRUARY 2016

Observation of Gravitational Waves from a Binary Black Hole Merger

B. P. Abbott *et al.*[*]
(LIGO Scientific Collaboration and Virgo Collaboration)
(Received 21 January 2016; published 11 February 2016)

On September 14, 2015 at 09:50:45 UTC the two detectors of the Laser Interferometer Gravitational-Wave Observatory simultaneously observed a transient gravitational-wave signal. The signal sweeps upwards in frequency from 35 to 250 Hz with a peak gravitational-wave strain of 1.0×10^{-21}. It matches the waveform predicted by general relativity for the inspiral and merger of a pair of black holes and the ringdown of the resulting single black hole. The signal was observed with a matched-filter signal-to-noise ratio of 24 and a false alarm rate estimated to be less than 1 event per 203 000 years, equivalent to a significance greater than 5.1σ. The source lies at a luminosity distance of 410^{+160}_{-180} Mpc corresponding to a redshift $z = 0.09^{+0.03}_{-0.04}$. In the source frame, the initial black hole masses are $36^{+5}_{-4} M_\odot$ and $29^{+4}_{-4} M_\odot$, and the final black hole mass is $62^{+4}_{-4} M_\odot$, with $3.0^{+0.5}_{-0.5} M_\odot c^2$ radiated in gravitational waves. All uncertainties define 90% credible intervals. These observations demonstrate the existence of binary stellar-mass black hole systems. This is the first direct detection of gravitational waves and the first observation of a binary black hole merger.

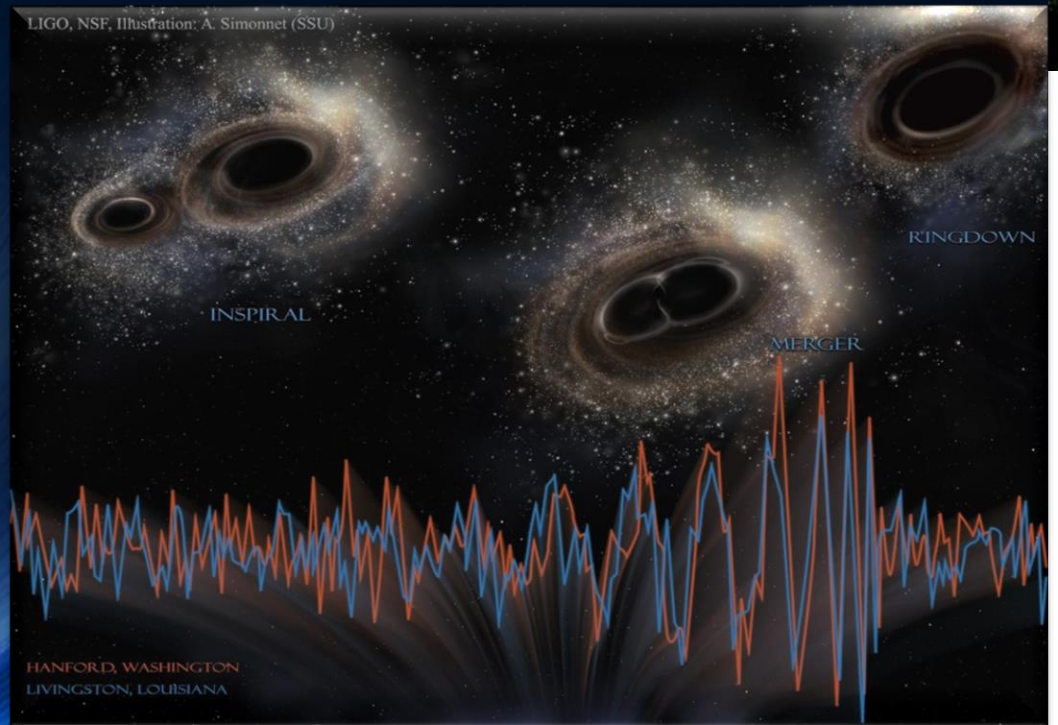
1. Direkter Nachweis von Gravitationswellen
Signalform: Verschmelzung von zwei **schwarzen** Löchern

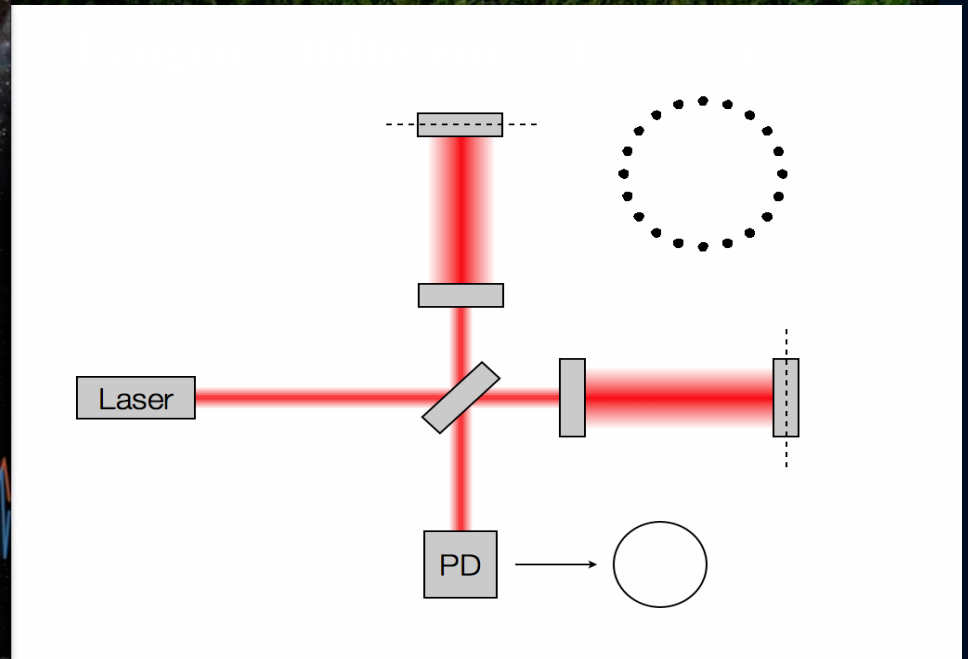
Gravitationswellen gefunden: LIGO!!!

Kollision zweier Schwarzer Löcher GW150914

Massen: 36 & 29 Sonnenmassen

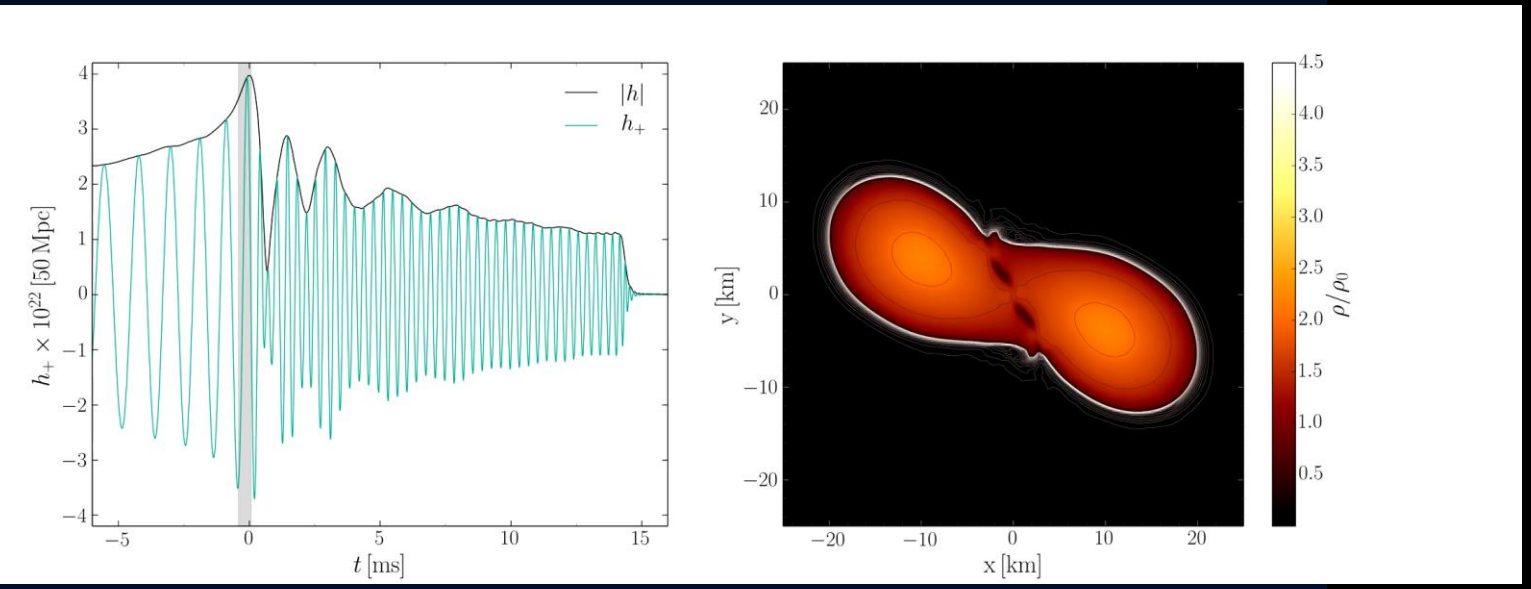
Abstand zur Erde 410 Mpc (1.34 Milliarden Lichtjahre)





Computersimulationen mit dem Einstein-Toolkit

In diesem Teil wird ein Einblick in die allgemein-relativistische Simulation auf Supercomputern gegeben. Unter Zuhilfenahme des Einstein-Toolkits werden unterschiedliche, realistische Systeme betrachtet (z.B. Neutronenstern-Kollisionen mit Aussendung von Gravitationswellen)



Studium & Promotion
+ Studieninteressierte
+ Studierende
+ Physik-Lernzentrum
+ Praktika
+ Anfängerpraktikum
+ Fortgeschrittenen Praktikum
+ Promotion
+ Akademische Feiern
+ Dozentinnen und Dozenten

Physikalisches Praktikum für Fortgeschrittene

Bitte beachten Sie, dass Sie für die Teilnahme am Forschungs- und Laborpraktikum im ***Bachelor- oder Masterstudiengang Physik*** oder ***Lehramt Physik*** eingeschrieben sein müssen. Voraussetzung für die Teilnahme ist ein abgeschlossenes Anfängerpraktikum.

Die Online-Anmeldung zum **Wintersemester 2022/23** für alle **Fortgeschrittenen-Praktika** ist in der Zeit vom **12.09.2022 bis zum 09.10.2022** freigeschaltet.

Anmeldungen zum Fortgeschrittenen-Praktikum WISe 2022/23

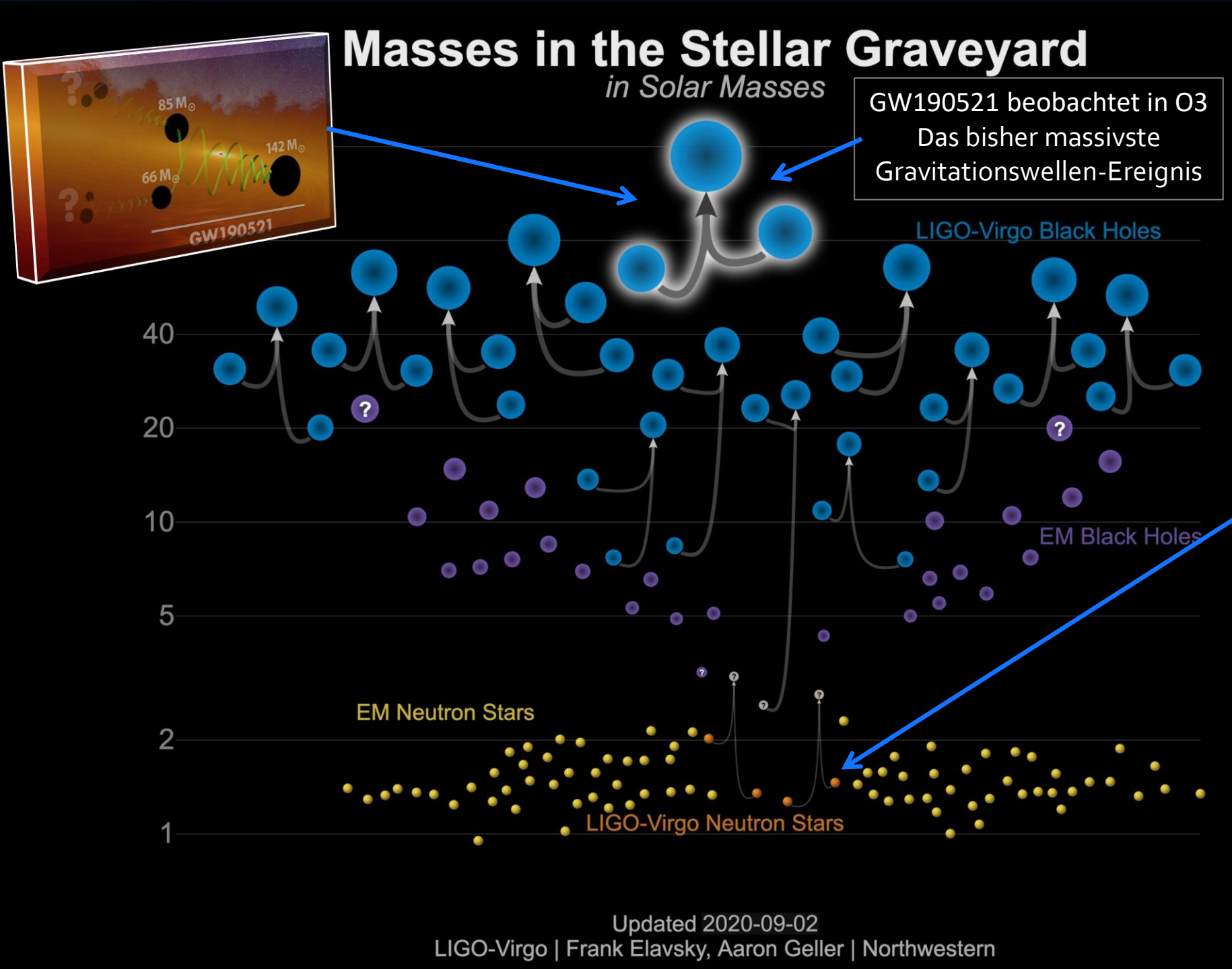
(Die Anmeldung für das SoSe 2022 ist abgeschlossen!)

Schnelleinstieg

- Informationssystem QIS / LSF
- Dekanat
- Prüfungsamt
- Bibliothek
- Vorlesungsverzeichnis
- Fachschaft Physik
- Gleichstellungsrat Physik
- Antidiskriminierungsstelle

Detektierte Gravitationswellen

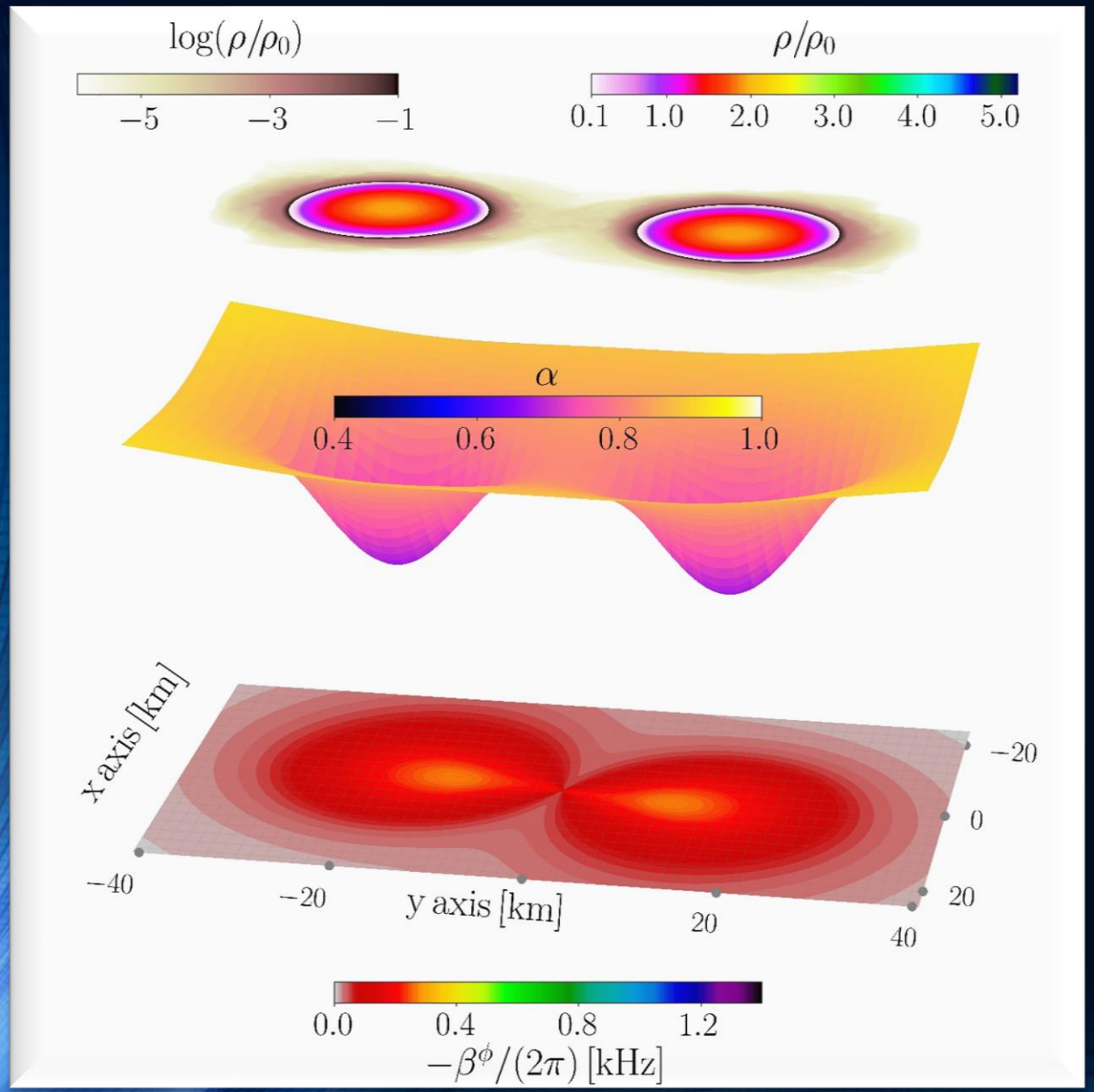
In den ersten beiden Beobachtungsläufen (O1+O2) konnten 11 Gravitationswellen detektiert werden, wobei einer dieser Gravitationswellen (GW170817) durch die Kollision zweier Neutronensterne verursacht wurde welche sich vor ungefähr 130 Millionen Jahren ereignete.

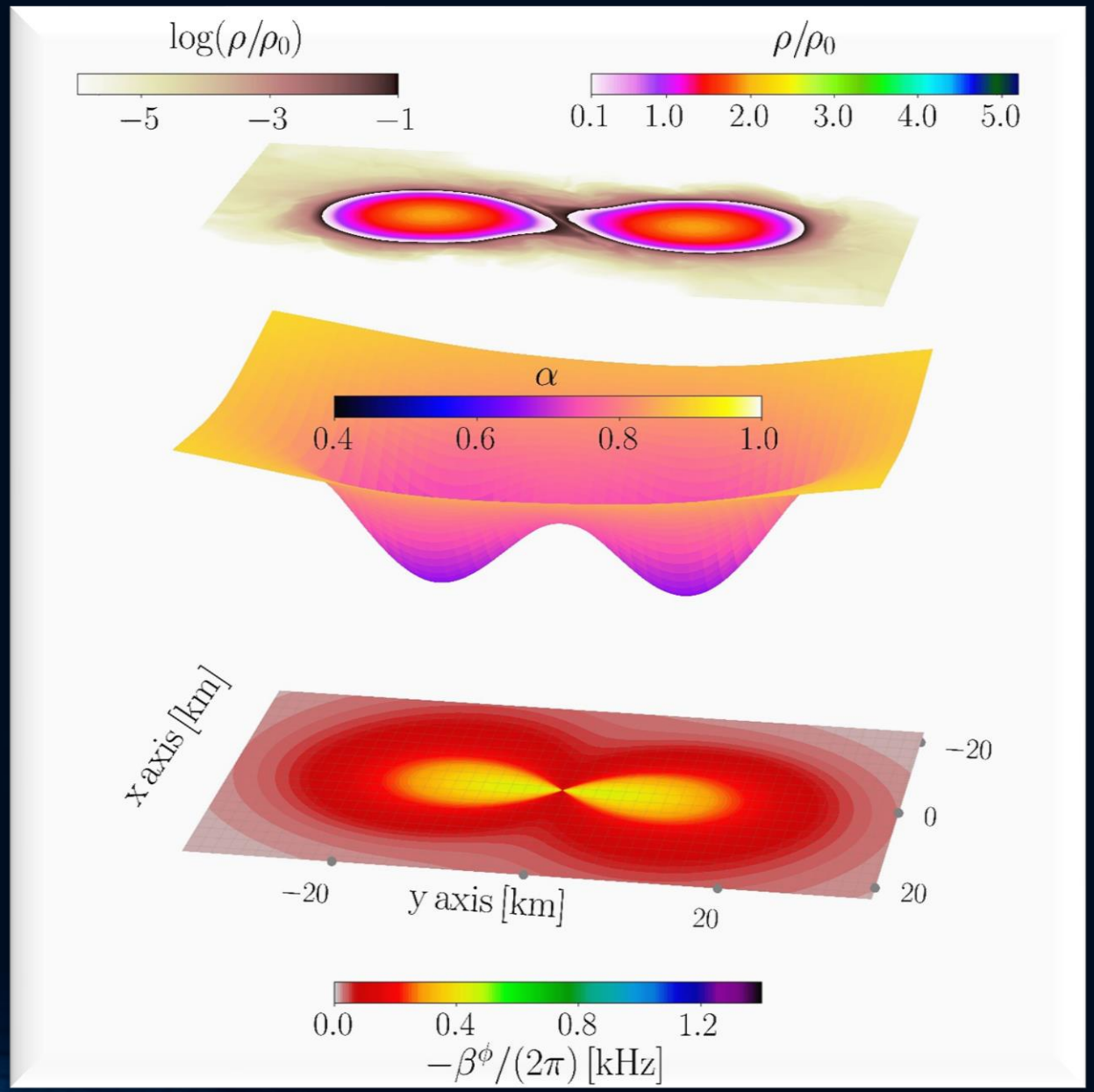


The long-awaited event GW170817

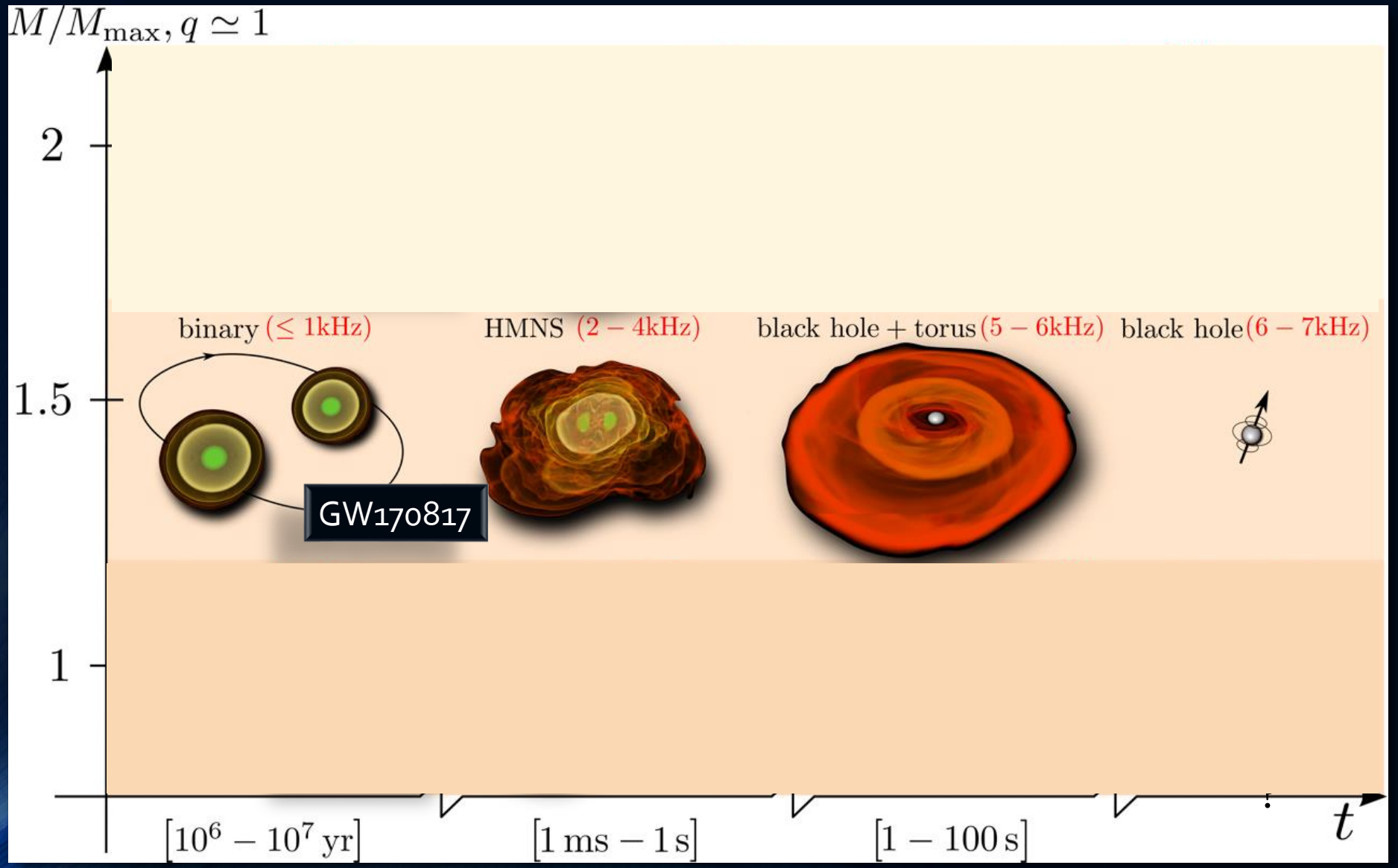
	Low-spin priors ($\lvert\chi\rvert \leq 0.05$)	High-spin priors ($\lvert\chi\rvert \leq 0.89$)
Primary mass m_1	1.36–1.60 $M_\odot$	1.36–2.26 $M_\odot$
Secondary mass m_2	1.17–1.36 $M_\odot$	0.86–1.36 $M_\odot$
Chirp mass $\mathcal{M}$	$1.188^{+0.004}_{-0.002} M_\odot$	$1.188^{+0.004}_{-0.002} M_\odot$
Mass ratio m_2/m_1	0.7–1.0	0.4–1.0
Total mass $m_{\rm tot}$	$2.74^{+0.04}_{-0.01} M_\odot$	$2.82^{+0.47}_{-0.09} M_\odot$
Radiated energy $E_{\rm rad}$	$> 0.025 M_\odot c^2$	$> 0.025 M_\odot c^2$
Luminosity distance $D_{\rm L}$	40^{+8}_{-14} Mpc	40^{+8}_{-14} Mpc
Viewing angle Θ	$\leq 55°$	$\leq 56°$
Using NGC 4993 location	$\leq 28°$	$\leq 28°$
Combined dimensionless tidal deformability $\tilde{\Lambda}$	≤ 800	≤ 700
Dimensionless tidal deformability $\Lambda(1.4 M_\odot)$	≤ 800	≤ 1400

The late inspiral phase (density, lapse and shift)





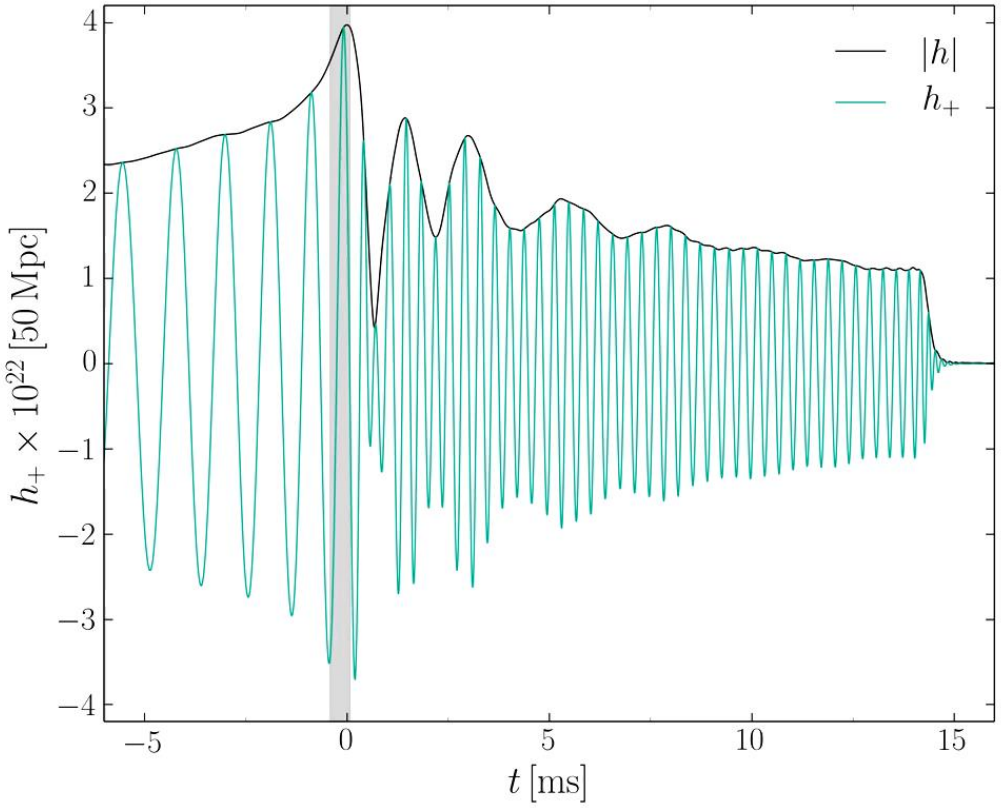
Broadbrush picture

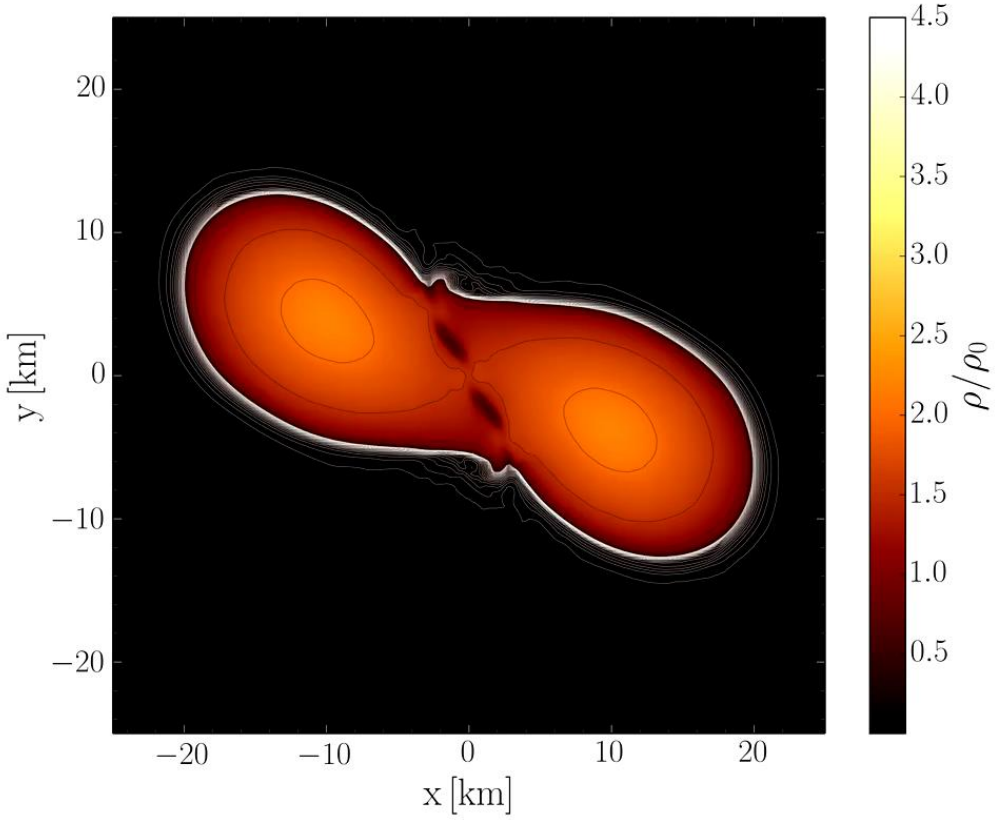


Gravitational Waves and Hypermassive Hybrid Stars

ALF2-EOS: Mixed phase region starts at $3\rho_0$ (see red curve), initial NS mass: 1.35 M_{solar}

Hanauske, et.al. PRD, 96(4), 043004 (2017)





Gravitational wave amplitude
at a distance of 50 Mpc

Rest mass density distribution $\rho(x,y)$
in the equatorial plane

in units of the nuclear matter density ρ_0

Jupyter Notebook

Neutronenstern Kollisionen mit dem Einstein Toolkit

Allgemeine Relativitätstheorie n

General Theory of Relativity on t

Vorlesung gehalten an der J.W.Goethe 2021)

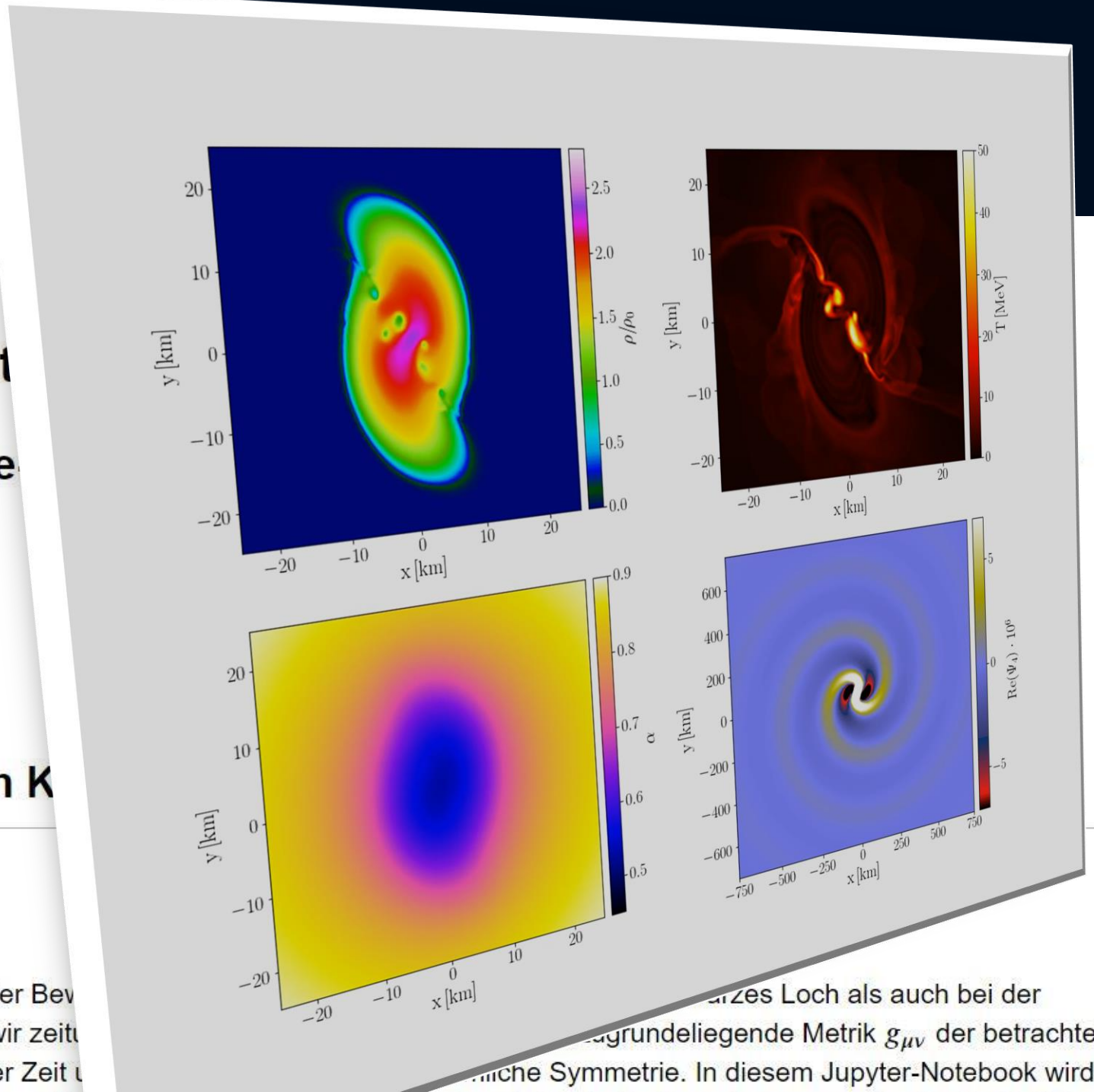
von Dr.phil.nat. Dr.rer.pol. Matthias Hanauske

Frankfurt am Main 17.06.2021

Dritter Vorlesungsteil: Neutronenstern K

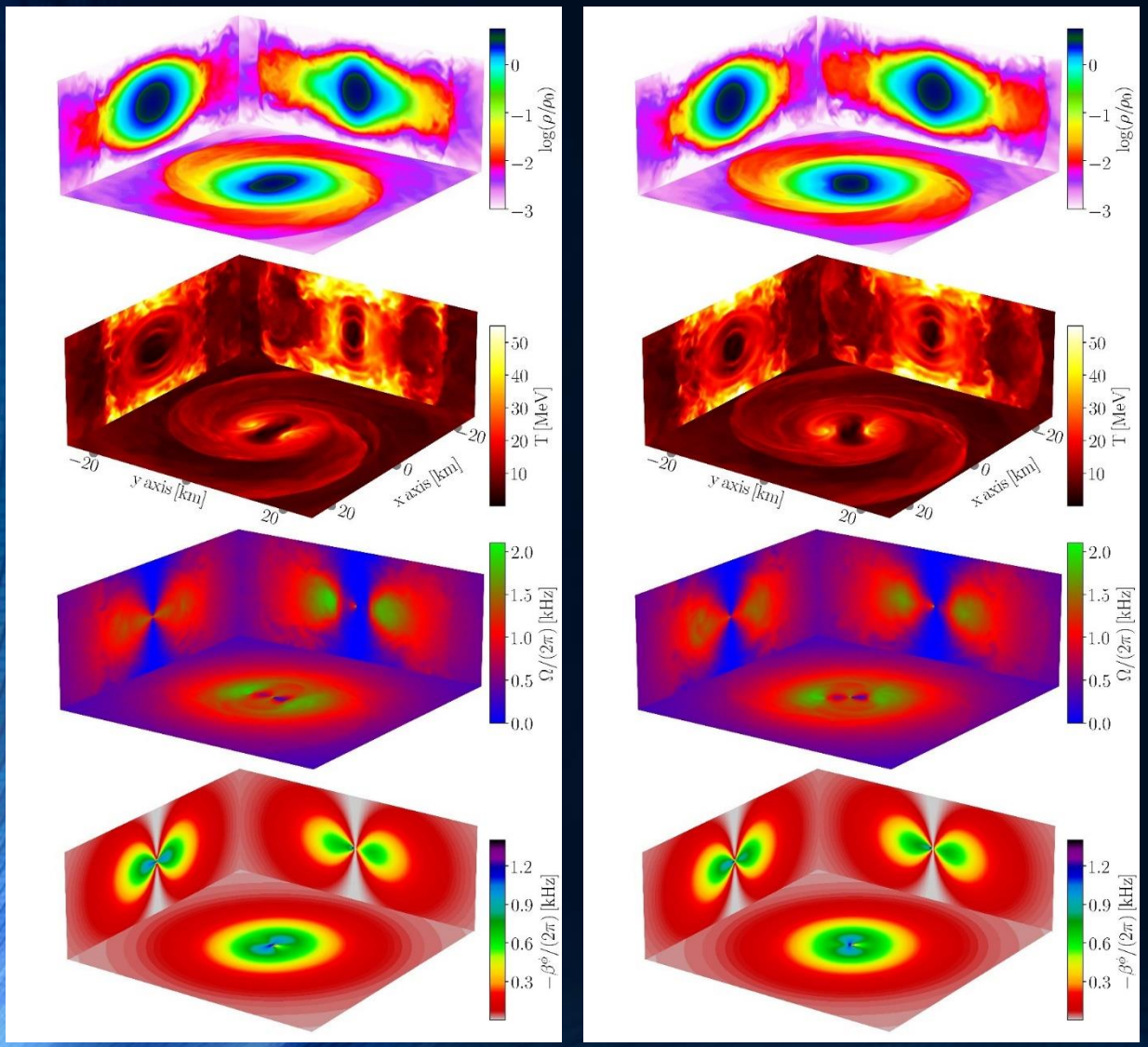
Einführung in den (3+1)-Split

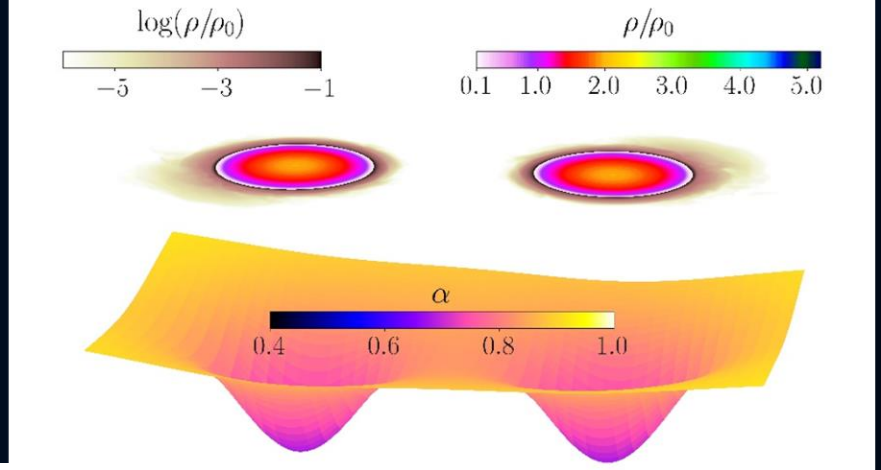
In den bisherigen Jupyter-Notebooks, sowohl bei der Analyse der Bev
Berechnung der Eigenschaften von Neutronensternen, hatten wir zeitu
4-dimensionale Mannigfaltigkeit $\mathcal{M}$ veränderte sich nicht mit der Zeit u

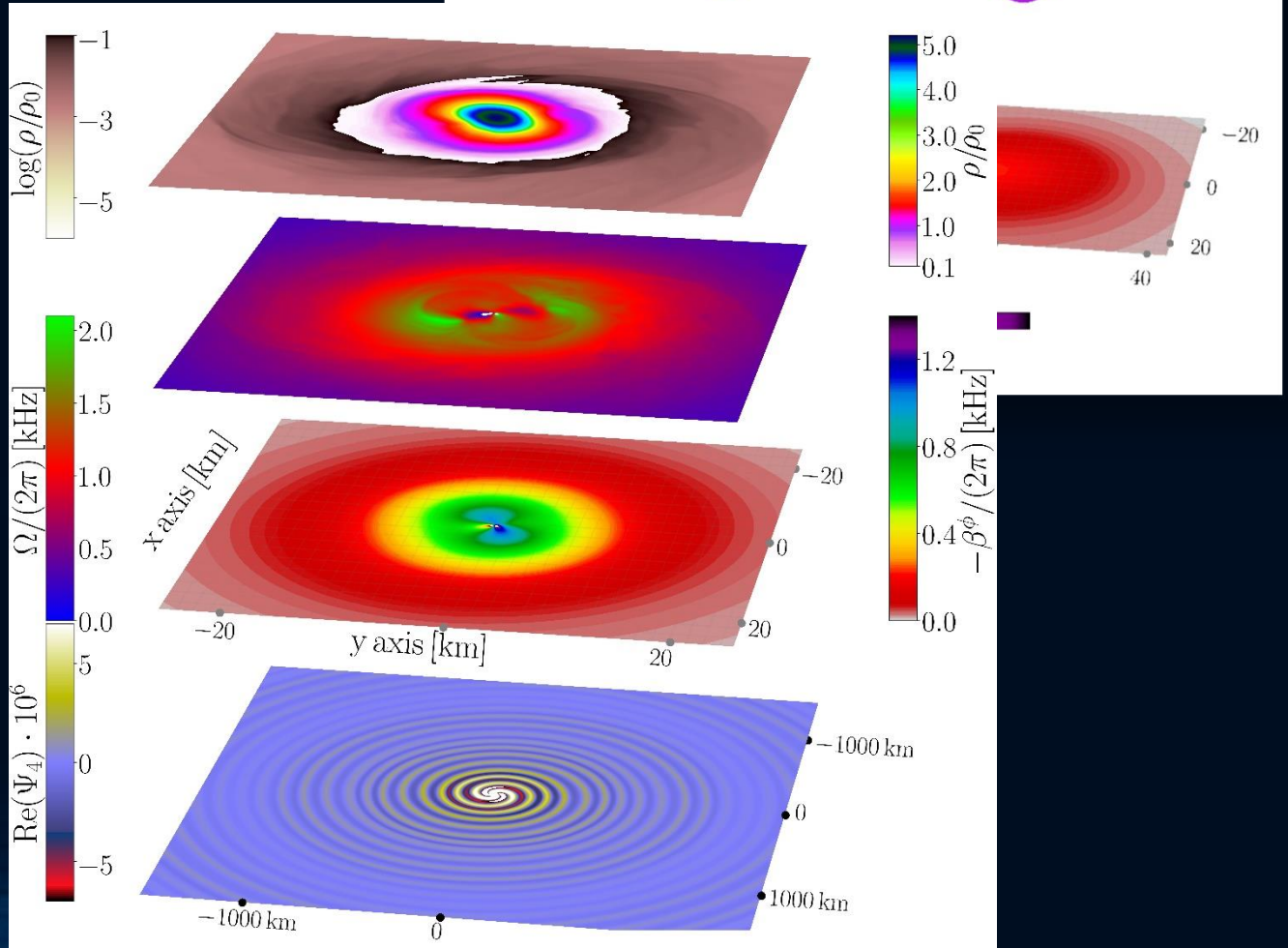


...rzes Loch als auch bei der
...grundeliegende Metrik $g_{\mu\nu}$ der betrachteten
...liche Symmetrie. In diesem Jupyter-Notebook wird

Neutronenstern Kollisionen mit dem Einstein Toolkit



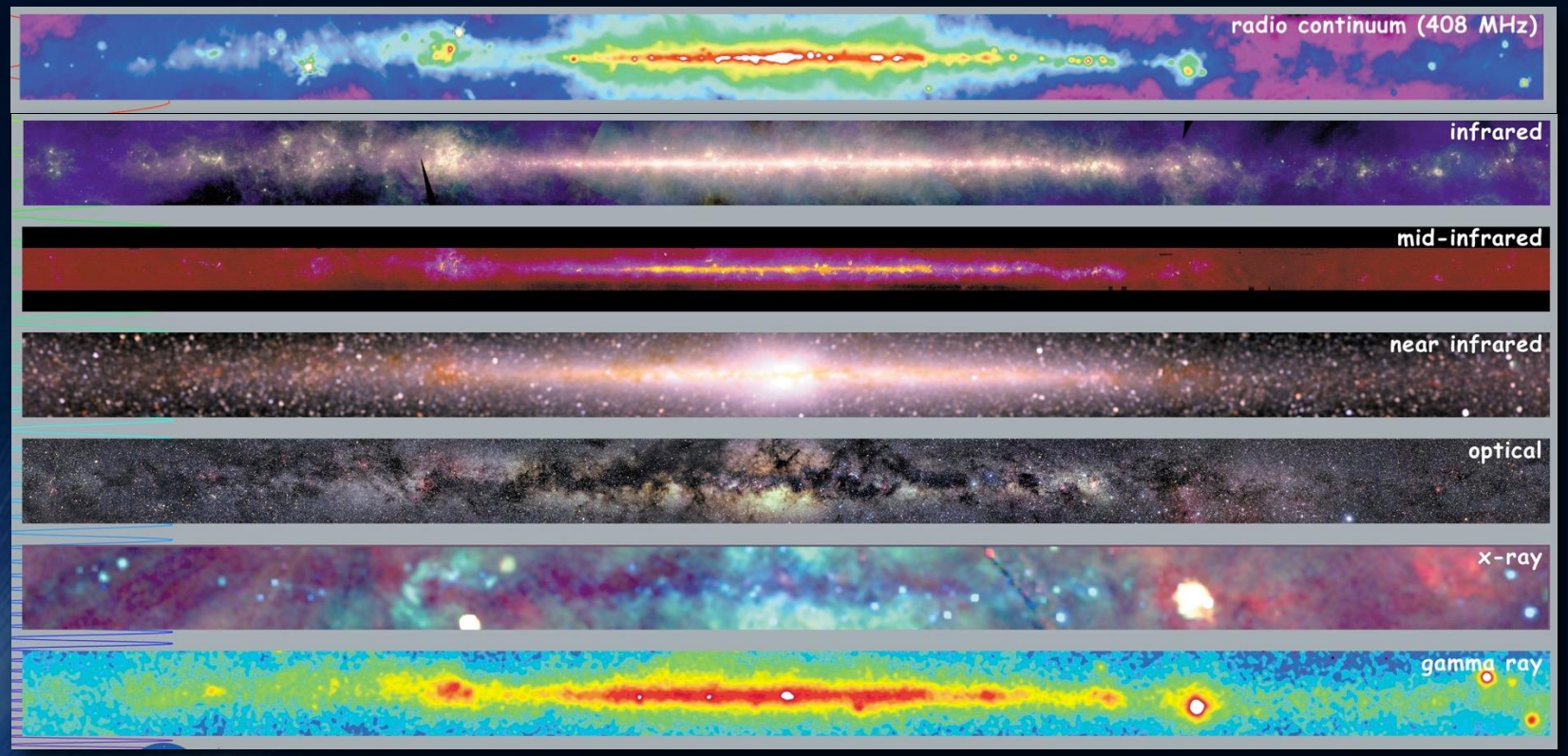




Die neue Art unser Universum zu betrachten

Lange Zeit über war das Studium von astrophysikalischen Vorgängen auf den mit den Augen sichtbaren Bereich limitiert und optische Teleskope entwickelten sich erst ab dem 16. Jahrhundert. Die Wahrnehmung des Universums in den anderen Frequenzbereichen der elektromagnetischen Strahlung wurde durch die Radio, Infrared and X-ray telescopes möglich und entwickelte sich erst im 20. Jahrhundert.

GSFC/NASA



Erkenntnisse mittels elektromagnetische Strahlung

Radio

w-IR

m-IR

n-IR

Optisch

Röntgen

Gamma

Gravitations-wellen

Es ist so als ob die Menschheit eine neue wundersame Brille hat, ein neues Sinnesorgan, mit denen sie zuvor unbeobachtbare Ereignisse in unserem Universum wahrnehmen kann.

Physik der sozio-ökonomischen Systeme mit dem Computer (Online) *von* Dr.phil.nat.Dr.rer.pol. Matthias Hanauske

Nächster Zoom Link am 22.10.2021, 15:00-17:00 Uhr:
ID: ... ..., PWD: .....

Vorwort

Die Vorlesung *Physik der sozio-ökonomischen Systeme mit dem Computer* wurde im Wintersemester 2015/16 das erste Mal gehalten und viele der auf dieser Hauptseite erreichbaren Internetseiten basieren grundsätzlich auf dem damals erstellten Kurs. Das nebenstehende Video (ist in Arbeit!) gibt einen kurzen Überblick der Inhalte der Vorlesung. In der ersten Vorlesung (siehe Zoom Link in der rechten oberen Ecke) werden die Voraussetzungen besprochen, die man benötigt um einen benoteten bzw. unbenoteten Schein mit fünf Creditpoints zu erhalten.

Weiterführende Links

- Zoom Meeting Software
- Online-Lernplatform OLAT
- Online-Lernplatform Lon Capa

Physik der sozio-ökonomischen Systeme mit dem Computer (Physics of Socio-Economic Systems with the Computer) Vorlesung WS 2021/22

Auch in diesem Semester findet die Vorlesung nur Online statt!

Diese Internetseite fasst die Online-Angebote der Vorlesung *Physik der sozio-ökonomischen Systeme mit dem Computer* zusammen. Auf der linken Seite finden Sie die einzelnen Vorlesungspräsentationen (pdf-Dateien), Computerprogramme und weiterführende Links. Die Vorlesungstermine (Zoom Meetings, synchrones Lehrangebot) finden jeweils freitags von 15.00-17.00 Uhr statt. An den Online-Übungen können Sie entweder freitags vor (13.30-15.00 Uhr) oder nach der Vorlesung (17:00-18:30 Uhr) teilnehmen (Beginn der Online-Übungen erst am 29.10.2021). Alle Lehrangebote werden mittel der Zoom Meeting Software gemacht und die jeweiligen Zoom-Links sind in der rechten oberen Ecke dieser Internetseite angegeben.
Die Inhalte der Vorlesung gliedern sich in drei Teile (Teil I, Teil II, Teil III), die Sie in der zweiten oberen Spalte einsehen können. Weiteres Zusatzmaterial und diverse Online-Aufgaben sind über die Online-Lernplattformen OLAT und Lon Capa erhältlich (siehe *E-Learning*).

Weiterführende Literatur

- Schlee, Walter, Einführung in die Spieltheorie, Vieweg 2004
- Hofbauer, Josef, and Karl Sigmund. Evolutionary games and population dynamics. Cambridge university press, 1998
- Martin A. Nowak, Evolutionary Dynamics - Exploring the Equations of Life, 2006
- Albert-Laszlo Barabasi, Network science, Cambridge university press, 2016
- Matthias Hanauske, Evolutionäre Quanten-Spieltheorie im Kontext sozio-ökonomischer Systeme, 2011
- Vorlesungsmaterialien: Neue Entwicklungen in der Evolutionären Spieltheorie (2009)
- Vorlesungsmaterialien: Hochschul-Sommerkurs Money, Money, Money: Deutschlands Wirtschafts- und Finanzleben (2011)

Einführung

Key Question

How can one theoretically describe the time dependent evolution of the strategic behavior of an entire group of decision makers?

Theoretical Models used to answer the question:

(Evolutionary) Game Theory

[von Neumann 1928, Nash 1950, Smith 1972, Weibull 1997, Szabó/Fáth 07]

Theory of complex networks

[Barabasi/Albert 02, Mendes/Dorogovtsev 02, Jackson 10]

I.1.1 Definition eines Spiels

Die formale mathematische Definition eines *Simultanen (N Spieler)-(m Strategien) Spiels in strategischer Form mit Auszahlung* (siehe z.B. [2,3]) benötigt lediglich die Angabe dreier Größen: Die Menge $\mathcal{I}$ der Spieler, die Menge (der Raum) $\mathcal{S}$ der Strategien der Spieler und ihre Auszahlungsfunktion (Präferenzordnungen) $\$$.

Ein Spiel $\Gamma := (\mathcal{I}, \mathcal{S}, \$)$ in strategischer Form mit Auszahlung ist hinreichend definiert, wenn die folgenden drei Größen bekannt sind:

- Menge der Spieler: $\mathcal{I} = \{1, 2, \ldots, N\}$
  Die Menge der Spieler $\mathcal{I}$ kann unter Umständen aus unterschiedlichen Teilmengen bestehen, die ihrerseits unterschiedliche Strategiemengen $\mathcal{S}$ besitzen. In sozio-ökonomischen Netzwerken stellen die Spieler die jeweiligen Knoten des Netzwerkes dar (näheres siehe Teil II).
- Menge der reinen Strategien der Spieler: $\mathcal{S} = \mathcal{S}^1 \times \mathcal{S}^2 \times \ldots \times \mathcal{S}^N$
  Jeder Spieler $\mu \in \mathcal{I}$ besitzt eine eigene Menge an reinen Strategien $\mathcal{S}^\mu = \left\{s_1^\mu, s_2^\mu, \ldots, s_{m_\mu}^\mu\right\}$, wobei jede dieser m_μ Strategien eine für ihn mögliche Entscheidung darstellt.
- Präferenzordnungen der Spieler, quantifiziert durch eine vektorwertige Auszahlungsfunktion:
  $\$ = \left(\$^1, \$^2, \ldots, \$^N\right) : \mathcal{S} \to \mathbb{R}^N$
  Nachdem jeder Spieler (ohne die Entscheidung seiner Mitspieler zu kennen) eine Strategie aus seiner Strategienmenge $\mathcal{S}^\mu$ ausgewählt hat, beurteilt er die entstehende Strategienkombination $\mathcal{S}$ entsprechend seiner Präferenzordnung (Auszahlungsfunktion) $\$^\mu$.

Um diese formale Definition im einzelnen zu erklären, beschränken sich die folgenden Darlegungen auf den einfachsten Fall des simultanen

Physik der sozio-ökonomischen Systeme *mit dem Computer*

JOHANN WOLFGANG GOETHE UNIVERSITÄT
22.10.2021

MATTHIAS HANAUSKE

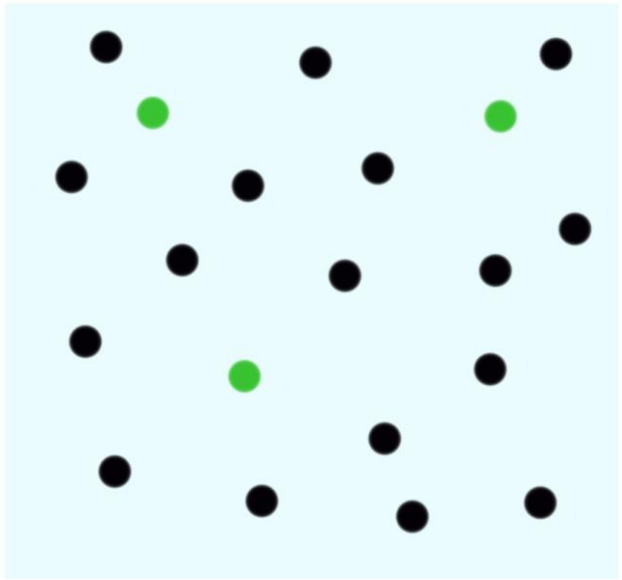
FRANKFURT INSTITUTE FOR ADVANCED STUDIES
JOHANN WOLFGANG GOETHE UNIVERSITÄT
INSTITUT FÜR THEORETISCHE PHYSIK
ARBEITSGRUPPE RELATIVISTISCHE ASTROPHYSIK
D-60438 FRANKFURT AM MAIN
GERMANY

Auch in diesem Semester (WS 2021/22) findet die Vorlesung und die Übungstermine nur Online statt.

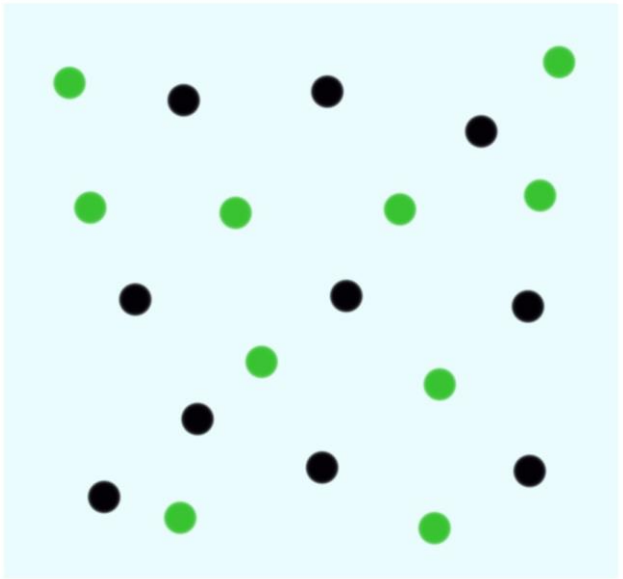
1. Vorlesung

Evolutionäre Spieltheorie

Die evolutionäre Spieltheorie betrachtet die zeitliche Entwicklung des strategischen Verhaltens einer gesamten Spielerpopulation.



zeitliche Entwicklung der Population



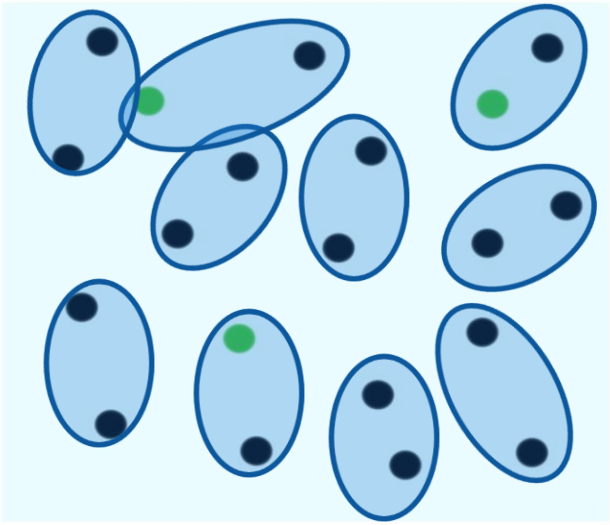
x(0)=0.15

x(10)=0.5

Mögliche Strategien: (grün , schwarz), Parameter t stellt die „Zeit“ dar.
x(t) : Anteil der Spieler, die im Zeitpunkt t die Strategie „grün“ spielen.

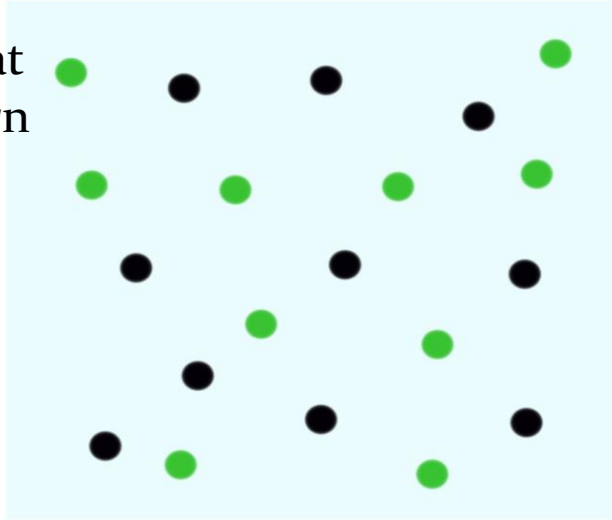
Evolutionäre Spieltheorie

Die einzelnen Akteure innerhalb der betrachteten Population spielen ein andauernd sich wiederholendes Spiel miteinander, wobei sich jeweils zwei Spieler zufällig treffen, das Spiel spielen und danach zu dem nächsten Spielpartner wechseln .



x(0)=0.15

Die Anfangspopulation von Spielern spielt zum Zeitpunkt t=0 das erste Mal das Spiel. Die Spieler wählen im Mittel zu 15% die grüne Strategie.

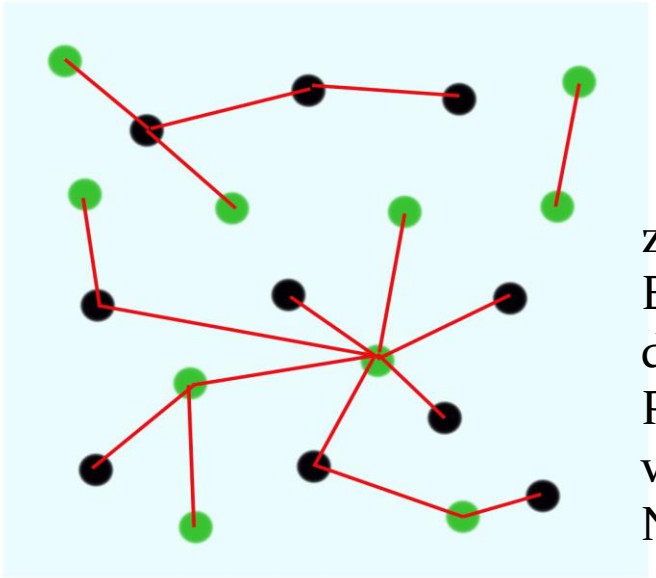


x(10)=0.5

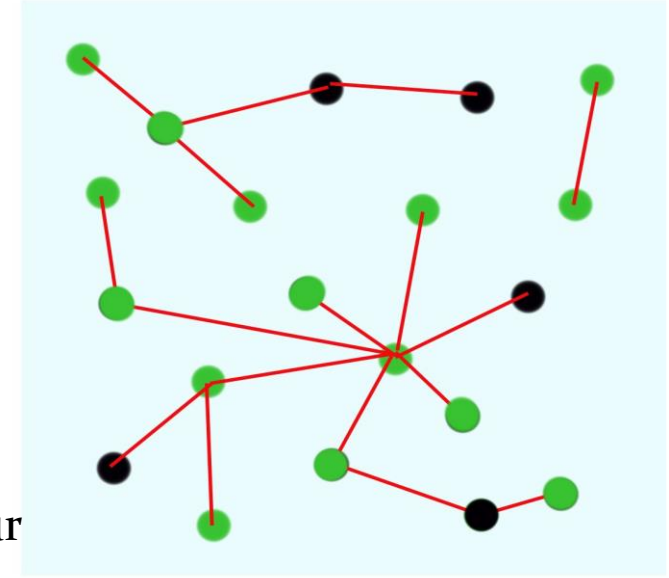
Das evolutionäre Spiel schreitet voran und die grüne Strategie wird für die Spieler zunehmend attraktiver. Zum Zeitpunkt t=10 spielen schon 50% grün.

Evolutionäre Spieltheorie auf komplexen Netzwerken

Viele in der Realität vorkommende evolutionäre Spiele werden auf einer definierten Netzwerkstruktur (Topologie) gespielt. Die Spieler der betrachteten Population sind hierbei nicht gleichwertig, sondern wählen als Spielpartner nur mit ihnen durch das Netzwerk verlinkte (verbundene) Partner aus.



zeitliche Entwicklung der Population auf vorgegebener Netzwerkstruktur



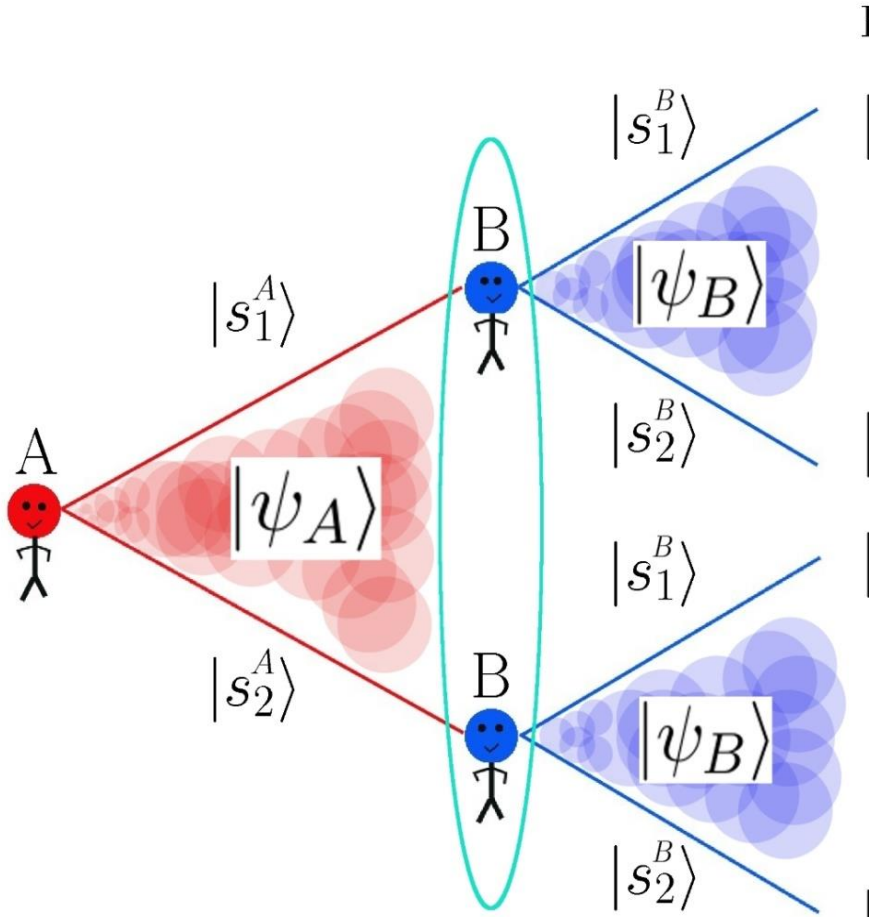
x(0)=0.5

x(10)=0.75

Mögliche Strategien: (grün , schwarz), Parameter t stellt die „Zeit“ dar.
x(t) : Anteil der Spieler, die im Zeitpunkt t die Strategie „grün“ spielen.
Die roten Verbindungslinien beschreiben die möglichen Spielpartner des Spielers

Quantum Games

The Quantum Game Tree



Basisvektoren

$|s_1^A\, s_1^B\rangle$

$|s_1^A\, s_2^B\rangle$

$|s_2^A\, s_1^B\rangle$

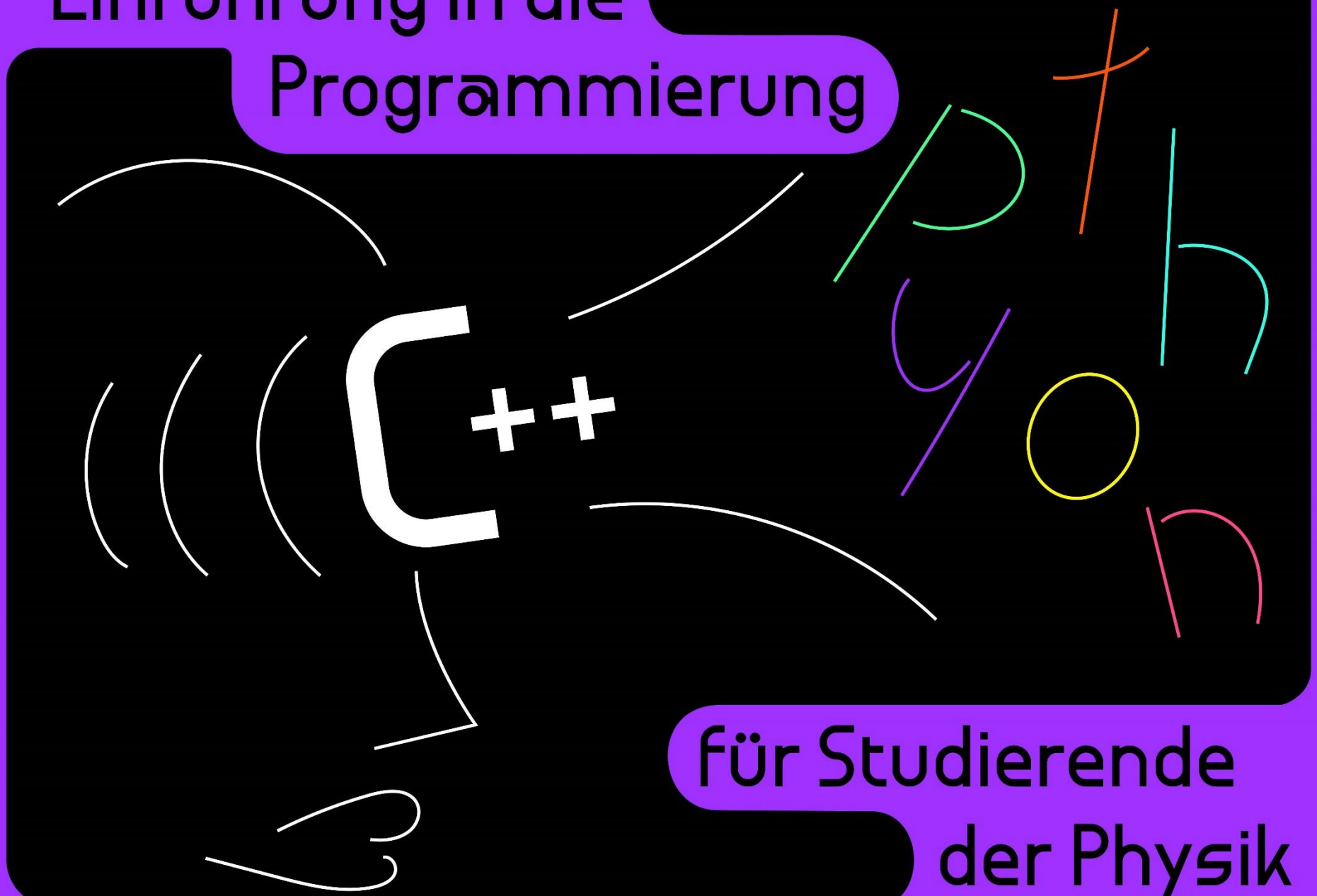
$|s_2^A\, s_2^B\rangle$

Zwei Spieler

Zustandsfunktion

$$|\Psi\rangle = \hat{\mathcal{J}}^\dagger \left(\hat{\mathcal{U}}_A \otimes \hat{\mathcal{U}}_B\right) \hat{\mathcal{J}}\ |s_1^A\, s_1^B\rangle$$

Einführung in die Programmierung



für Studierende der Physik