Allgemeine Relativitätstheorie mit dem Computer

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

ZOOM ONLINE MEETING
JOHANN WOLFGANG GOETHE UNIVERSITÄT
04. JUNI, 2021

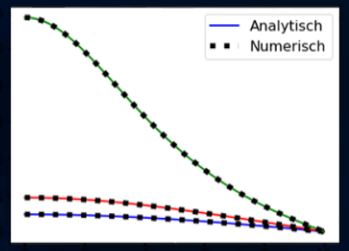
MATTHIAS HANAUSKE

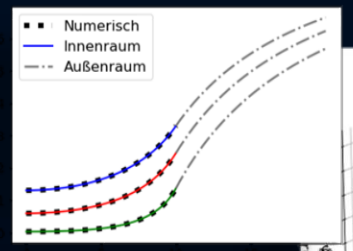
FRANKFURT INSTITUTE FOR ADVANCED STUDIES
JOHANN WOLFGANG GOETHE UNIVERSITÄT
INSTITUT FÜR THEORETISCHE PHYSIK
ARBEITSGRUPPE RELATIVISTISCHE ASTROPHYSIK
D-60438 FRANKFURT AM MAIN
GERMANY

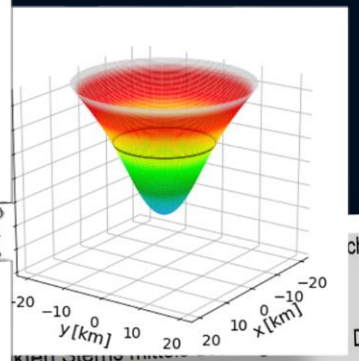
8. Vorlesung

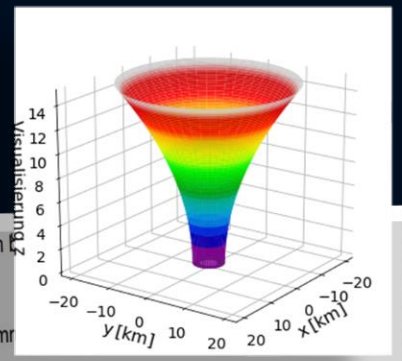
Jupyter Notebook

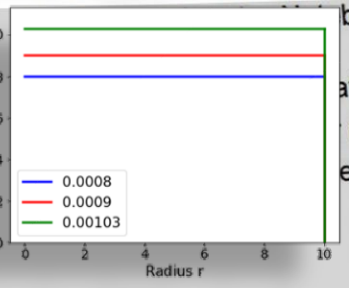
Die TOV-Gleichung: Zusätzliche Betrachtungen

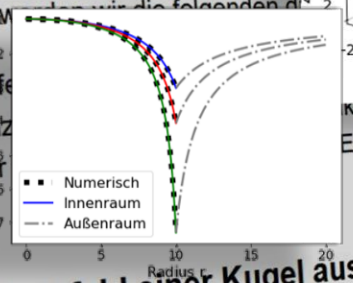












Euler-Verfahrens

Spezialfall: Gravitationsfeld einer Kugel aus inkompressibler Flüssigkeit (konstante Dichte)

Die TOV-Gleichungen wurden im Jahre 1939 gefunden und gründen auf den folgenden zwei Arbeiten: Tolman, Richard C. "Static solutions of Einstein's field equations for spheres of fluid." Physical Review 55.4 (1939): 364. und Oppenheimer, J. Robert, and George M. Volkoff. "On massive neutron cores." Physical Review 55.4 (1939): 374.. Es ist beachtlich, dass bereits im Jahre 1916, lange bevor die TOV-Gleichungen publiziert wurden, Karl Schwarzschild schon einen wichtigen Spezialfall der TOV-Gleichungen analytisch berechnete. Herr Schwarzschild betrachtete einen sphärisch symmetrischen Körper mit konstanter Dichte (siehe Karl Schwarzschild, "Gravitationsfeld einer Kugel aus inkompressibler Flüssigkeit", Sitzungsberichte der Königlich-Preussischen Akademie der Wissenschaften. Reimer, Berlin 1916, S:424-434) und berechnete die Eigenschaften dieses Körpers in der Einsteins allgemeiner Relativitätstheorie.

Im Folgenden werden wir auch diesen Spezialfall betrachten und die analytische Lösung mit unseren numerischen Berechnungen vergleichen. Wir starten hingegen nicht, wie Schwarzschild es damals, von der Einsteingleichung, sondern nehmen die TOV-Gleichungen als gegeben an.

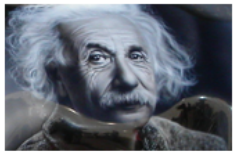
In [1]:
```
import numpy as np
from sympy import *
init_printing()
from einsteinpy.symbolic import *
```

GOETHE UNIVERSITÄT FRANKFURT AM MAIN

Startseite | Lehren & Lernen | Kursangebote | Allgemeine Relativität...

Allgemeine Relativitätstheorie mit dem Computer

- Allgemeine Relativitätstheorie mit dem Computer
  - Literaturverzeichnis
  - Einschreibung
  - Kursinhalt
  - Vorlesungsaufzeichnung
  - Aufgaben
  - Programme
    - Einführung in Jupyter Notebooks
    - Allgemeine Relativitätstheorie mit Python
    - Eigenschaften der Schwarzschild-Metrik
    - Radialer Fall eines Probekörpers in ein schwarzes Lo
    - Klassifizierung unterschiedlicher Bahnbewegungen
    - Der ISCO und die Photonensphäre
    - Maple Worksheets I
    - Das rotierende schwarzes Loch: Struktur der Horizon
    - Das rotierende schwarzes Loch: Klassifikation möglic
    - Maple Worksheets II
    - Die Tolman-Oppenheimer-Volkoff (TOV) Gleichung
    - Die TOV-Gleichung: Zusätzliche Betrachtungen
    - Jupyter Notebooks



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

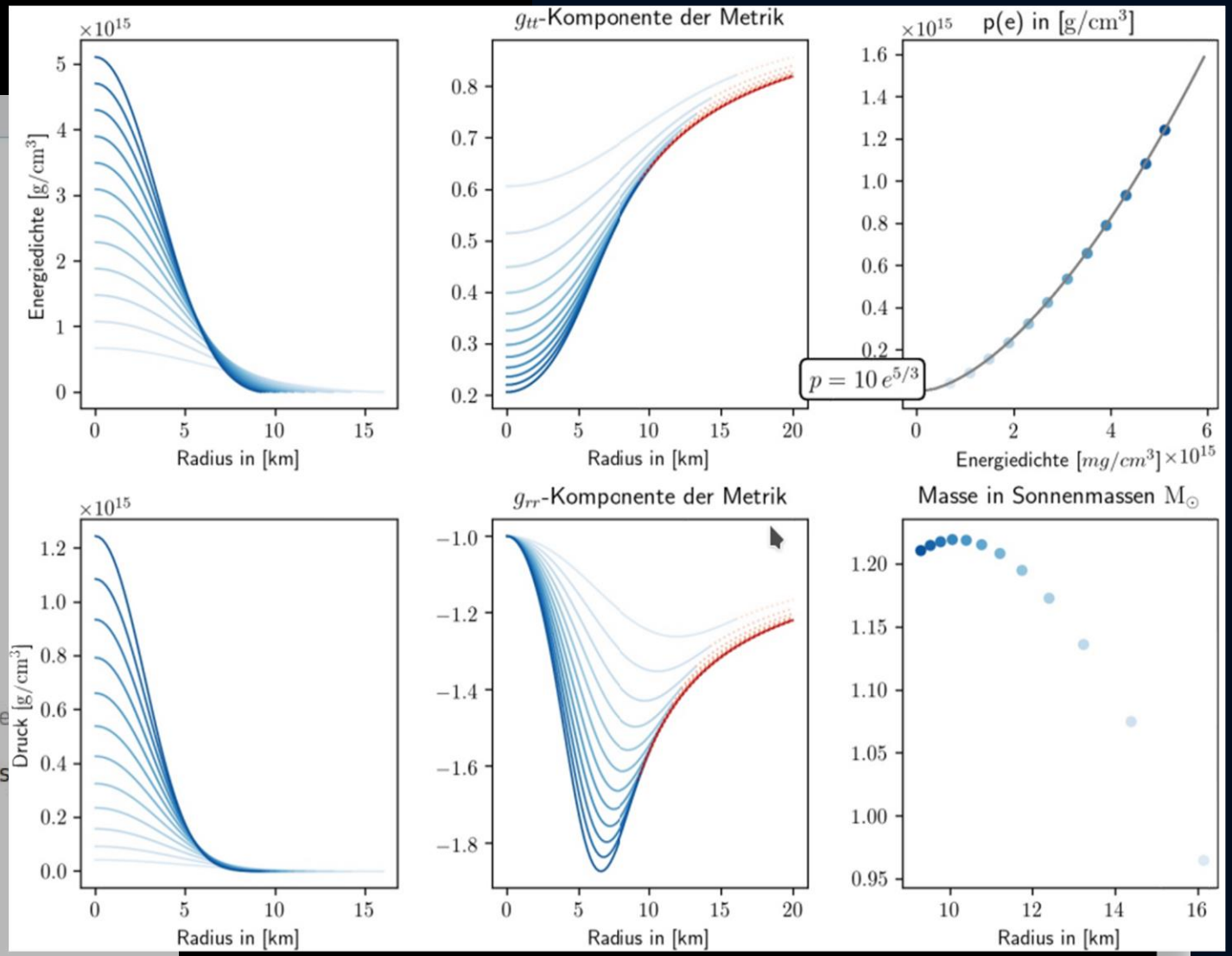
Vorlesung 7

Python-Programm

```
Datei  Bearbeiten  Ansicht  Lesezeichen  Einstellungen  Hilfe
(base) hanauske@hanauske-Aspire-A717-72G:~/VARTC/CoronaCampus2021/python/TOVPython$ python TOV-Sequence-plot2021.py
i = 0
Neutronensternradius [km]           = 16.125959999558688
Neutronensternmasse [Sonnenmassen]  = 0.9650804371812499
00-Metrikkomponente im Sternzentrum = 0.6065692130701107
```

TOV-Sequence-plot2021.py | VARTC2021_old8.html

```
import numpy as np
import matplotlib.pyplot as plt
import matplotlib
import math
import matplotlib.gridspec as gridspec

# Definition der Zustandsgleichung
K=10.0
Gamma=5.0/3.0
def eos(p):
    e=math.pow(p/K,1.0/Gamma)
    return e

# plot settings
params = {
    'figure.figsize'    : [10, 7.5],
    'text.usetex'       : True,
}
matplotlib.rcParams.update(params)

# Gitter zum Plotten von sechs unterschiedliche Diagramme definie
plt.figure(0)
gs = gridspec.GridSpec(2, 3, width_ratios=[1,1,1], wspace=0.3, hs
ax1 = plt.subplot(gs[0])
ax2 = plt.subplot(gs[1])
ax3 = plt.subplot(gs[2])
ax4 = plt.subplot(gs[3])
ax5 = plt.subplot(gs[4])
ax6 = plt.subplot(gs[5])
props = dict(boxstyle='round', facecolor='white', alpha=0.92)
```

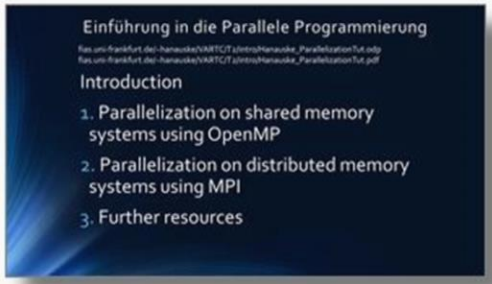



```
(base) hanauske@hanauske-Aspire-A717-72G:~/VARTC/CoronaCampus2021/python/TOVPython$ ffmpeg -framerate 1 -i './img-%0
3d.jpg' -s 2000x1500 TOV-Seq.mp4
```

TOVPython : bash

Einführung in die Parallele Programmierung

Wiederholung
siehe Vortragsfolien
Vorlesung 7

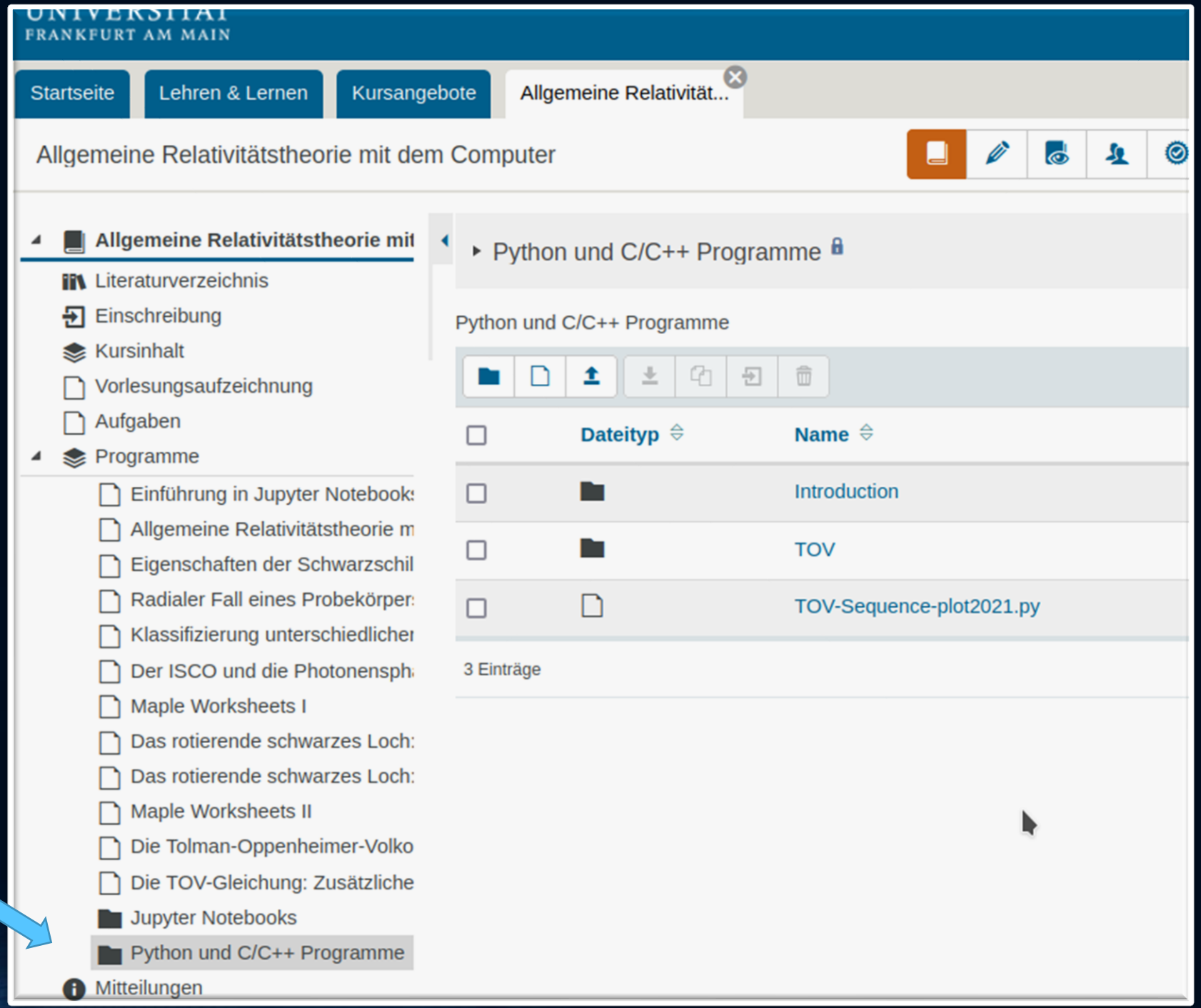


Paralleles Programmieren mit C++ und OpenMP/MPI

Computersimulationen von realistischen, komplizierten Problemen in der Allgemeinen Relativitätstheorie (z.B. die Simulation einer Neutronenstern Kollision) erfordern, sogar auf "Supercomputern", 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. Dieser Unterpunkt im Teil II der Vorlesung gibt eine Einführung in die parallele Programmierung mit C++ und OpenMP (Open Multi-Processing) / MPI (Message Passing Interface). Die Folien der Präsentation sind unter den folgenden Links einsehbar: Einführung in die parallele Programmierung (LibreOffice Datei , PDF Datei. Am Beispiel eines einfachen numerischen Problems (der Integration einer Funktion), wird das Programmierparadigma der parallelen Programmierung erläutert. Zunächst wird ein einfaches sequentielles C++-Programm erstellt, das die Integration der Funktion $f(x) = \frac{1}{1+a\,x^2}$ in den Grenzen [0,1] und den Werten $a \in [0, 10]$ mit dem Gauß'schen Integrationsverfahren numerisch berechnet (siehe sequentielle Version 1 und sequentielle Version 2). In dem Programm wird der Wert $W(a)$ des Integrals $\int_0^1 \frac{1}{1+a\,x^2}\,dx$ für die Parameterwerte $a \in [0, 10]$ ausgegeben und mit dem analytischen Ergebnis $W(a) = \frac{\arctan(\sqrt{a})}{\sqrt{a}}$ verglichen. Die Parallelisierung dieses sequentiellen Programms wird zunächst mit OpenMP (siehe OpenMP Version 1 , OpenMP Version 2 , OpenMP Version 3 , OpenMP Version 4 und OpenMP Version 5) und danach mit MPI (siehe MPI Version 1 , MPI Version 2) durchgeführt.

Python und C/C++ Programme

Auf der OLAT Seite des Kurses finden Sie die im folgenden besprochenen C/C++ Programme



8. Vorlesung

Vorlesung 8

In der vorigen Vorlesung hatten wir in das parallele Programmieren mit C++ und OpenMP/MPI eingeführt und am Beispiel eines einfachen numerischen Problems (die Integration einer Funktion) die grundlegende Vorgehensweise eines OpenMP und MPI Programms kennengelernt. In dieser Vorlesung werden wir das numerische Lösen der Tolman-Oppenheimer-Volkoff (TOV Gleichungen, siehe Vorlesungen 6 und 7) mittels des Eulerverfahrens in einem C++ Programm durchführen und die besprochenen Parallelisierungsparadigmen (OpenMP und MPI) anwenden. Es wird sowohl ein sequentielles C++ Programm zur Berechnung der Eigenschaften von Neutronensternen, als auch eine mit OpenMP und MPI parallelisierte Version besprochen (siehe auch Teil II: Paralleles Programmieren mit C++ und OpenMP/MPI).

Sequentielles C++ Programm zur Berechnung der Eigenschaften von Neutronensternen durch numerisches Lösen der TOV Gleichungen mittels des Eulerverfahrens

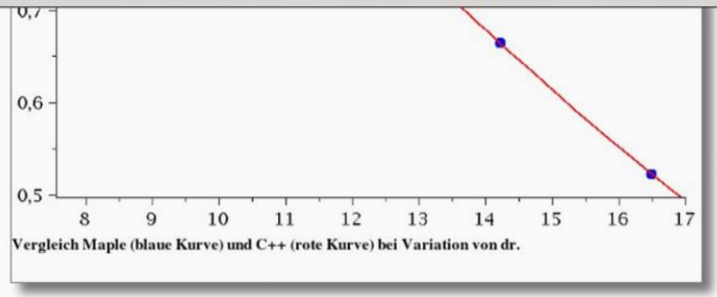
Ausgehend von der, in der Vorlesung 6 hergeleiteten TOV Gleichung, wird mittels des einfachen Euler-Verfahrens die Differentialgleichung in C++ implementiert (siehe TOV sequentielle Version 2.1)). Die numerisch berechneten Werte des Neutronensternradius und seiner gravitativen Masse werden am Ende des Programms im Terminal ausgegeben. Die berechneten Resultate der Druck- und Energiedichtenprofile kann man mit den in der Vorlesung 6/7 mit Python (bzw. Maple) berechneten Ergebnissen vergleichen (siehe TOV sequentielle Version 2.1) mit Ausgabe der Ergebnisse in ein Textfile und Jupyter Notebook Die TOV-Gleichung: Zusätzliche Betrachtungen und Vergleich mit C++ Ergebnissen). Man kann zusätzlich, auf sequenzielle Weise, mehrere Neutronensterne bei festgelegter Zustandsgleichung berechnen. Dies wird einfach realisiert, indem man eine weitere for-Schleife über die zentrale Energiedichte einbaut. Der Hauptunterschied zur vorigen sequentiellen Version ist, das nun eine ganze Sequenz von Neutronensternen berechnet wird und man sich somit die Masse-Radius-Relation bei gegebener Zustandsgleichung vorstellen kann. Zusätzlich zur Version 2.1) werden auch noch die

Vorlesung 8

In dieser Vorlesung werden wir uns mit dem Programmierparadigma der parallelen Programmierung befassen. Wie schon am Ende der vorigen Vorlesung erwähnt, ist es bei der Konzeption von rechenintensiven Computerprogrammen wichtig, dass die Rechenleistung des Computers stets voll ausgelastet ist und separate, voneinander unabhängige Teilaufgaben innerhalb der Programme möglichst gleichzeitig (parallel) berechnet werden. Da die meisten der großen Computerprogramme im Bereich der Allgemeinen Relativitätstheorie in der Programmiersprache C/C++ geschrieben sind, werden wir die parallele Programmierung im Folgenden am Beispiel dieser Programmiersprache verdeutlichen. 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.

Die Programmiersprachen C/C++ und Python unterscheiden sich voneinander und bei der Erstellung der Quelldateien (source codes) eines C++ Programms ist einiges zu beachten. Jede Variable, die im Programm benutzt wir muss zunächst mit einem Typ deklariert werden (z.B. "int" für eine ganze Zahl, oder "double" für eine Fließkommazahl), mit dem Präprozessorbefehl "#include" bindet man benötigte "Header-Dateien" in das Programm ein (ähnlich den Python Bibliotheken/Modulen), die Strukturierung eines C++ Programms benötigt nicht die in Python verwendete Block-Einrückung (z.B. bei for-Schleifen), sondern verwendet geschweifte Klammern und in C/C++ muss man den Quelltext mittels eines Kompilierers in ein ausführbares Programm umwandeln. Die Art und Weise wie das parallele Programm zu konzipieren ist, hängt

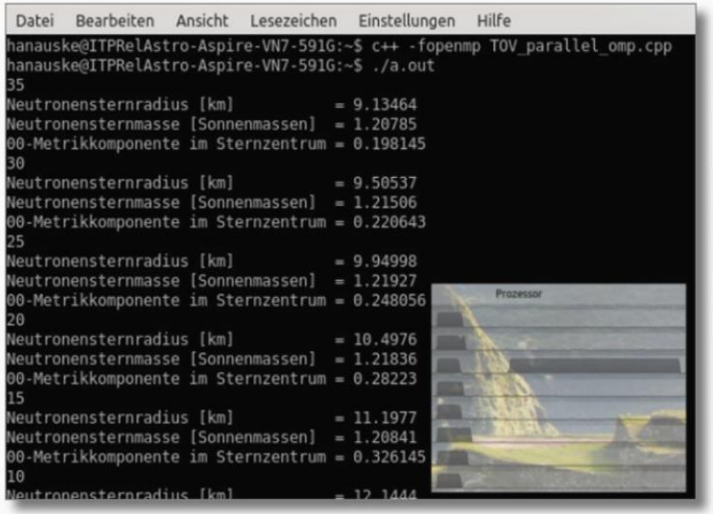
Parallele Programme siehe Teil 2 der Internetseite der Vorlesung



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.

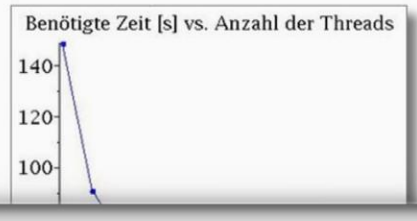


2.4) Parallele OpenMP-Version 2 (2.3) mit geordneter Terminal-Ausgabe)

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

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

Teil II: Parallele OpenMP - Version des TOV-Programms 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, γ)= $(P/K)^{1/\gamma}$) für Neutronensterne und Weiße Zwerge bzw. (e(P,Bag)=3 p + 4 Bag) für Quarksterne im MIT-Bag Model). Wichtig ist, 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, dass es (in Abhängigkeit wieviele CPU-Kerne man in seinem Computer hat), viel schneller läuft.

Struktur des parallelen OpenMP-C++ Programms

```
#include <iostream>                  //Ein-/Ausgabe (Include-Dateien)
#include <math.h>                    //Mathematisches
#include <omp.h>                     //OpenMP
#include <stdio.h>                   //Fuer die Ausgabedatei

//Definition einer polytropen Zustandsgleichung
double eos(double p, double K, double gamma)
  {
    double e;
    e=pow(p/K,1.0/gamma);
```

Das parallele openMP Programm


```
#include <iostream>                  //Ein-/Ausgabe (Include-Dateien)
#include <math.h>                    //Mathematisches
#include <omp.h>                     //OpenMP
#include <stdio.h>                   //Fuer die Ausgabedatei

//Definition einer polytropen Zustandsgleichung
double eos(double p, double K, double gamma)
  {
    double e;
    e=pow(p/K,1.0/gamma);
    return e;
  }

//Definition einer MIT-Bag Zustandsgleichung mit cs^2=1/3 (Ueberladen der Funktion eos(...))
double eos(double p, double Bag)
  {
    double e;
    e=3.0*p + 4.0*Bag;
    return e;
  }

main(void)  //Hauptprogramm
  {
    //Variablendeklarationen
    int i,anz=150;
    double M,p,e,r,nu,dM,dp,de,dr,dnu,dec;
    double Er[anz],EM[anz],Enu[anz],Eec[anz];
    double K,gamma;
    double Bag;

    //Ausgabedatei
    FILE *ausgabe;
    ausgabe = fopen("output/tov.txt", "w+");          //"tov.txt" im Unterverzeichnis "output" öffnen zum speichern der Ergebnisse

    //Variableninitialisierung
    dr=0.00001;
    dec=0.00005;
    gamma=5.0/3.0;
    K=7.3015389;
    Bag=0.0001339578066;                              //Entspricht B^(1/4)=170 MeV
```

Das parallele openMP Programm

An dieser Stelle des Programms beginnt die parallelisierte Schleife. Es werden, abhängig von der Anzahl der verfügbaren Prozessoren im Computer, mehrere Threads erzeugt, die gleichzeitig die einzelnen Aufgaben der Schleife ausführen. Das OpenMP-Pragma '*#pragma omp parallel for private(i,M,p,e,r,nu,dM,dp,de,dnu)*' vor der for-Schleife realisiert die Parallelisierung, wobei der private(...) Zusatz sicherstellt, dass die während der Berechnung benötigten Hilfsvariablen (z.B. M, dM) nicht von anderen Threads überschrieben werden. Jeder Thread greift sich einen der 150 (anz=150) zu berechnenden Neutronensterne raus, löst die TOV-Gleichung und berechnet die Masse, den Radius und die für die zentrale g_{00}-Komponente nötige Größe des Sterns. Wenn ein Thread fertig mit der Berechnung ist, nimmt er sich den nächsten noch nicht berechneten Stern vor.

Kleiner Fehler!
i muss nicht “private” sein

```
    //for Schleife zur Berechnung mehrerer Sterne
#pragma omp parallel for private(i,M,p,e,r,nu,dM,dp,de,dnu)
    for (i=0;i<anz;i++)
    {
      M=0;
      r=pow(10,-14);
      p=K*pow((i+1)*dec,gamma);                    //Fuer polytrope Zustandsgleichung
//      p=1.0/3.0*(i+1)*dec-4.0/3.0*Bag;             //Fuer MIT-Bag Zustandsgleichung
      nu=0;
      Eec[i]=eos(p,K,gamma);                       //Fuer polytrope Zustandsgleichung
//      Eec[i]=eos(p,Bag);                           //Fuer MIT-Bag Zustandsgleichung

      //do-while Schleife (Numerische Lösung der TOV-Gleichung)
      do
        {
          e=eos(p,K,gamma);                        //Wert der Energiedichte bei momentanen Druck (polytrope Zustandsgleichung)
//          e=eos(p,Bag);                            //Wert der Energiedichte bei momentanen Druck (MIT-Bag Zustandsgleichung)
          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)
          dnu=(M + 4*M_PI*r*r*r*p)/(r*(r-2*M))*dr;      //Metrikzunahme bei momentanem r und Schrittweite dr
```

Das parallele openMP Programm

Kleiner Fehler!
i muss nicht “private” sein

```
    //for Schleife zur Berechnung mehrerer Sterne
#pragma omp parallel for private(i,M,p,e,r,nu,dM,dp,de,dnu)
    for (i=0;i<anz;i++)
    {
      M=0;
      r=pow(10,-14);
      p=K*pow((i+1)*dec,gamma);                          //Fuer polytrope Zustandsgleichung
//      p=1.0/3.0*(i+1)*dec-4.0/3.0*Bag;                   //Fuer MIT-Bag Zustandsgleichung
      nu=0;
      Eec[i]=eos(p,K,gamma);                             //Fuer polytrope Zustandsgleichung
//      Eec[i]=eos(p,Bag);                                 //Fuer MIT-Bag Zustandsgleichung

      //do-while Schleife (Numerische Lösung der TOV-Gleichung)
      do
        {
          e=eos(p,K,gamma);                              //Wert der Energiedichte bei momentanen Druck (polytrope Zustandsgleichung)
//          e=eos(p,Bag);                                  //Wert der Energiedichte bei momentanen Druck (MIT-Bag Zustandsgleichung)
          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)
          dnu=(M + 4*M_PI*r*r*r*p)/(r*(r-2*M))*dr;       //Metrikzunahme bei momentanem r und Schrittweite dr
          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
          nu=nu+dnu;                                     //momentane Metrik des Neutronensterns innerhalb des Radius r
        }
        while(p>0);

      Er[i]=r;
      EM[i]=M;
      Enu[i]=log(1-2*M/r)/2-nu;
    }
```

Das parallele openMP Programm

Nach der parallelen Ausführung der Schleife arbeitet wieder nur ein Prozessorkern, der das restliche Programm ausführt (Ausgabe der von den unterschiedlichen Threads berechneten Ergebnisse in eine externe Datei).

```
  //Geordnete Ausgabe der Masse, des Radius und der zentralen g00-Metrikkomponente in die Ausgabedatei
  fprintf(ausgabe,"# R[km]    M[Msol]   g00    ec[MeV/fm3] \n");
  for (i=0;i<anz;i++)
  {
    fprintf(ausgabe, "%f %f %f %e \n",Er[i],EM[i]/1.4766,exp(2*Enu[i]),Eec[i]*pow(10,6)/1.3234);
  }

  fclose(ausgabe);                                    //Ausgabedatei schliessen

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

Das parallele MPI - Programm

Teil II: Parallele MPI - Version des TOV-Programms mit geordneter Ausgabe in eine Datei und variabler Zustandsgleichung

Diese Version entspricht der OpenMP-Version Version 2.6), benutzt jedoch MPI und nicht OpenMP zur Parallelisierung und kann somit auch auf heutigen Großrechneranlagen, die über eine hohe Anzahl von Rechenknoten mit einer Vielzahl von CPU-Kernen verfügen, parallel ausgeführt werden. Wichtig ist nun, dass man das Programm mit dem folgenden Befehl compiliert: 'mpic++ TOV_parallel_omp2_eos_time.cpp' und das Programm mit 'mpirun -np 6 ./a.out' ausführt ('-np 6' ist hier nur ein Beispiel und die Zahl 6 gibt die Anzahl der Prozesse an).

Struktur des parallelen MPI-C++ Programms

```
#include <iostream>                    //Ein-/Ausgabe (Include-Dateien)
#include <math.h>                      //Mathematisches
#include <stdio.h>                     //Fuer die Ausgabedatei
#include <mpi.h>                       //MPI

//Definition einer polytropen Zustandsgleichung
```

Das parallele MPI - Programm


```
#include <iostream>                    //Ein-/Ausgabe (Include-Dateien)
#include <math.h>                      //Mathematisches
#include <stdio.h>                     //Fuer die Ausgabedatei
#include <mpi.h>                       //MPI

//Definition einer polytropen Zustandsgleichung
double eos(double p, double K, double gamma)
  {
    double e;
    e=pow(p/K,1.0/gamma);
    return e;
  }

//Definition einer MIT-Bag Zustandsgleichung mit cs^2=1/3 (Ueberladen der Funktion eos(...))
double eos(double p, double Bag)
  {
    double e;
    e=3.0*p + 4.0*Bag;
    return e;
  }

main( int nArguments, char **arguments )  //Hauptprogramm
  {
    MPI::Status status;
    MPI::Init(nArguments,arguments);
    int psize=MPI::COMM_WORLD.Get_size();
    int id=MPI::COMM_WORLD.Get_rank();
    printf("Prozess id= %i \n",id);
```

Das parallele MPI - Programm

Beim Ausführen des Programms spezifiziert der User mit wie vielen Prozessen er das Programm ausführen will (z.B. mit sechs Prozessen 'mpirun -np 6 ./a.out'). Ab diesem Zeitpunkt läuft das Programm mit sechs Prozessen, denen man im Laufe der weiteren Programmabfolge unterschiedliche Aufgaben zuweisen sollte, damit sie nicht alle das gleiche Ausführen. Die in der letzten Zeile angegebene Terminalausgabe erfolgt z.B. bei sechs Prozessen sechs mal; die Variable 'id' bezeichnet hier die fortlaufende Nummer des Prozesses (0,1,..,5).

```
//Variablendeklarationen
int i,anz=150;
double M,p,e,r,nu,dM,dp,de,dr,dnu,dec;
double Er[anz],EM[anz],Enu[anz],Eec[anz];
double K,gamma;
double Bag;

//Variableninitialisierung
dr=0.00001;
dec=0.00005;
gamma=5.0/3.0;
K=7.3015389;
Bag=0.0001339578066;                          //Entspricht B^(1/4)=170 MeV

//for Schleife zur Berechnung mehrerer Sterne
```

Das parallele MPI - Programm


```
//Variablendeklarationen
int i,anz=150;
double M,p,e,r,nu,dM,dp,de,dr,dnu,dec;
double Er[anz],EM[anz],Enu[anz],Eec[anz];
double K,gamma;
double Bag;

//Variableninitialisierung
dr=0.00001;
dec=0.00005;
gamma=5.0/3.0;
K=7.3015389;
Bag=0.0001339578066;                                    //Entspricht B^(1/4)=170 MeV

//for Schleife zur Berechnung mehrerer Sterne
//Aufteilung der zu berechnenden Sterne auf die einzelnen MPI-Prozesse
for (i=id;i<anz;i=i+psize)
{
```

Das parallele MPI - Programm

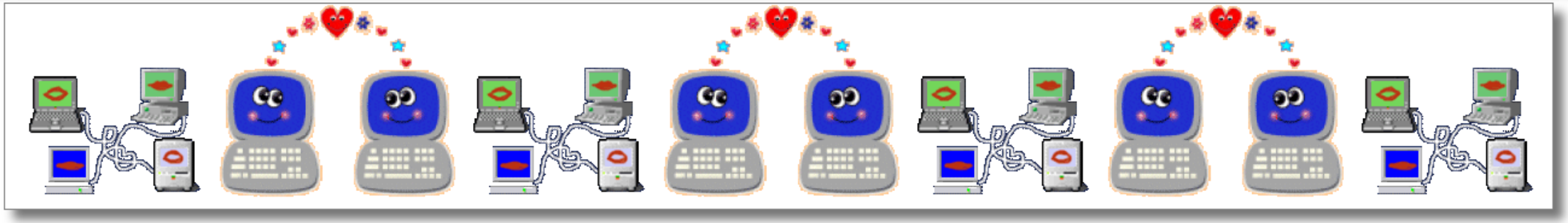
Eine sinnvolle Aufteilung der einzelnen Aufgaben des TOV-Programms auf die jeweiligen Prozesse kann man z.B. realisieren, indem die 150 (anz=150) zu berechnenden Neutronensterne auf die jeweiligen Prozesse aufteilt werden. In dem die for-Schleife von einem prozessabhängigen Startwert anfängt und in Schrittweiten 'psize' (Anzahl der Prozesse, hier z.B. 6) geht, rechnet jeder Prozess einen anderen Stern aus. Prozess 'id=0' rechnet z.B. die Sterne 'i=0,6,12,18,..' und Prozess 'id=4' rechnet z.B. die Sterne 'i=4,10,16,22,..' aus. Im Gegensatz zu den Threads in der OpenMP-Version wissen die einzelnen Prozesse der MPI Version nichts über die Berechnungen und Ergebnisse der anderen Prozesse, so dass die Werte der berechneten Ergebnisse übermittelt werden müssen - dies geschiet mit einem 'MPI::COMM_WORLD.Send(...)'-Kommando. In dieser Version senden alle Prozesse ihre Ergebnisse an Prozess mit 'id=0'.

```
    //for Schleife zur Berechnung mehrerer Sterne
    //Aufteilung der zu berechnenden Sterne auf die einzelnen MPI-Prozesse
    for (i=id;i<anz;i=i+psize)
    {
      M=0;
      r=pow(10,-14);
      p=K*pow((i+1)*dec,gamma);                    //Fuer polytrope Zustandsgleichung
//      p=1.0/3.0*(i+1)*dec-4.0/3.0*Bag;             //Fuer MIT-Bag Zustandsgleichung
      nu=0;
      Eec[i]=eos(p,K,gamma);                       //Fuer polytrope Zustandsgleichung
//      Eec[i]=eos(p,Bag);                           //Fuer MIT-Bag Zustandsgleichung

      //do-while Schleife (Numerische Lösung der TOV-Gleichung)
      do
        {
          e=eos(p,K,gamma);                        //Wert der Energiedichte bei momentanen Druck (polytrope Zustandsgleichung)
//          e=eos(p,Bag);                            //Wert der Energiedichte bei momentanen Druck (MIT-Bag Zustandsgleichung)
          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)
          dnu=(M + 4*M_PI*r*r*r*p)/(r*(r-2*M))*dr;     //Metrikzunahme bei momentanem r und Schrittweite dr
          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
          nu=nu+dnu;                               //momentane Metrik des Neutronensterns innerhalb des Radius r
        }
        while(p>0);
```

Das parallele MPI - Programm


```
    //for Schleife zur Berechnung mehrerer Sterne
    //Aufteilung der zu berechnenden Sterne auf die einzelnen MPI-Prozesse
    for (i=id;i<anz;i=i+psize)
    {
      M=0;
      r=pow(10,-14);
      p=K*pow((i+1)*dec,gamma);                          //Fuer polytrope Zustandsgleichung
//      p=1.0/3.0*(i+1)*dec-4.0/3.0*Bag;                  //Fuer MIT-Bag Zustandsgleichung
      nu=0;
      Eec[i]=eos(p,K,gamma);                             //Fuer polytrope Zustandsgleichung
//      Eec[i]=eos(p,Bag);                                //Fuer MIT-Bag Zustandsgleichung

      //do-while Schleife (Numerische Lösung der TOV-Gleichung)
      do
        {
          e=eos(p,K,gamma);                              //Wert der Energiedichte bei momentanen Druck (polytrope Zustandsgleichung)
//          e=eos(p,Bag);                                 //Wert der Energiedichte bei momentanen Druck (MIT-Bag Zustandsgleichung)
          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)
          dnu=(M + 4*M_PI*r*r*r*p)/(r*(r-2*M))*dr;       //Metrikzunahme bei momentanem r und Schrittweite dr
          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
          nu=nu+dnu;                                     //momentane Metrik des Neutronensterns innerhalb des Radius r
        }
        while(p>0);

      Er[i]=r;
      EM[i]=M;
      Enu[i]=log(1-2*M/r)/2-nu;

      //Alle Prozesse (ausser Prozess 0) senden ihre berechneten Ergebnisse an Prozess 0
      if(id != 0)
      {
        MPI::COMM_WORLD.Send( &Er[i], 1, MPI::DOUBLE, 0, 1);
        MPI::COMM_WORLD.Send( &EM[i], 1, MPI::DOUBLE, 0, 1);
        MPI::COMM_WORLD.Send( &Enu[i], 1, MPI::DOUBLE, 0, 1);
        MPI::COMM_WORLD.Send( &Eec[i], 1, MPI::DOUBLE, 0, 1);
      }

    }
```

```
  //Alle Prozesse (ausser Prozess 0) senden ihre berechneten Ergebnisse an Prozess 0
  if(id != 0)
  {
    MPI::COMM_WORLD.Send( &Er[i], 1, MPI::DOUBLE, 0, 1);
    MPI::COMM_WORLD.Send( &EM[i], 1, MPI::DOUBLE, 0, 1);
    MPI::COMM_WORLD.Send( &Enu[i], 1, MPI::DOUBLE, 0, 1);
    MPI::COMM_WORLD.Send( &Eec[i], 1, MPI::DOUBLE, 0, 1);
  }

}
```



```
//Geordnete Ausgabe der Masse, des Radius, der zentralen g00-Metrikkomponente und der zentralen Energiedichte in die Ausgabedatei
//Die Ausgabe erfolgt nur von dem Prozess 0, der zunaechst alle berechneten und an ihn gesendeten Daten empfaengt
if (id==0)
{
  for (int proc=1;proc<psize;proc++)
  {
    for (i=proc;i<anz;i=i+psize)
      {
          MPI::COMM_WORLD.Recv( &Er[i], 1, MPI::DOUBLE, proc, 1, status );
          MPI::COMM_WORLD.Recv( &EM[i], 1, MPI::DOUBLE, proc, 1, status );
          MPI::COMM_WORLD.Recv( &Enu[i], 1, MPI::DOUBLE, proc, 1, status );
          MPI::COMM_WORLD.Recv( &Eec[i], 1, MPI::DOUBLE, proc, 1, status );
       }
  }
  //Ausgabedatei
  FILE *ausgabe;
  ausgabe = fopen("output/tov.txt", "w+");          //"tov.txt" im Unterverzeichnis "output" öffnen zum speichern der Ergebnisse
```

Das parallele MPI - Programm


```
  //Geordnete Ausgabe der Masse, des Radius, der zentralen g00-Metrikkomponente und der zentralen Energiedichte in die Ausgabedatei
  //Die Ausgabe erfolgt nur von dem Prozess 0, der zunaechst alle berechneten und an ihn gesendeten Daten empfaengt
  if (id==0)
  {
    for (int proc=1;proc<psize;proc++)
    {
      for (i=proc;i<anz;i=i+psize)
        {
            MPI::COMM_WORLD.Recv( &Er[i], 1, MPI::DOUBLE, proc, 1, status );
            MPI::COMM_WORLD.Recv( &EM[i], 1, MPI::DOUBLE, proc, 1, status );
            MPI::COMM_WORLD.Recv( &Enu[i], 1, MPI::DOUBLE, proc, 1, status );
            MPI::COMM_WORLD.Recv( &Eec[i], 1, MPI::DOUBLE, proc, 1, status );
         }
    }
    //Ausgabedatei
    FILE *ausgabe;
    ausgabe = fopen("output/tov.txt", "w+");          //"tov.txt" im Unterverzeichnis "output" öffnen zum speichern der Ergebnisse

    fprintf(ausgabe,"# R[km]    M[Msol]   g00    ec[MeV/fm3] \n");
    for (i=0;i<anz;i++)
    {
      fprintf(ausgabe, "%f %f %f %e \n",Er[i],EM[i]/1.4766,exp(2*Enu[i]),Eec[i]*pow(10,6)/1.3234);
    }

    fclose(ausgabe);                                  //Ausgabedatei schliessen
  }

  MPI::Finalize ( );
  return 0;                                         //main beenden (Programmende)
}
```

Das parallele MPI - Programm


```
    //Ausgabedatei
    FILE *ausgabe;
    ausgabe = fopen("output/tov.txt", "w+");          //"tov.txt" im Unterverzeichnis "output" öffnen zum speichern der Ergebnisse

    fprintf(ausgabe,"# R[km]    M[Msol]   g00    ec[MeV/fm3] \n");
    for (i=0;i<anz;i++)
    {
      fprintf(ausgabe, "%f %f %f %e \n",Er[i],EM[i]/1.4766,exp(2*Enu[i]),Eec[i]*pow(10,6)/1.3234);
    }

    fclose(ausgabe);                                  //Ausgabedatei schliessen
  }

  MPI::Finalize ( );
  return 0;                                         //main beenden (Programmende)
}
```

Prozess '0' empfängt dannach alle Daten und gibt diese in die Ausgabedatei aus.

Lon Capa Übungsaufgaben

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

Aufgaben im Kurs allgemeine Relativitätstheorie mit dem Computer

Aufgaben im Teil I: Analytische Berechnungen und numerische Simulationen in Maple

Berechnung von Christoffelsymbolen der Schwarzschild-Metrik

Berechnung des Riemann Tensors der Schwarzschild-Metrik

Probekörper fällt radial in ein nichtrotierendes schwarzes Loch

Geodätische Bewegung eines Probekörpers um ein nichtrotierendes schwarzes Loch

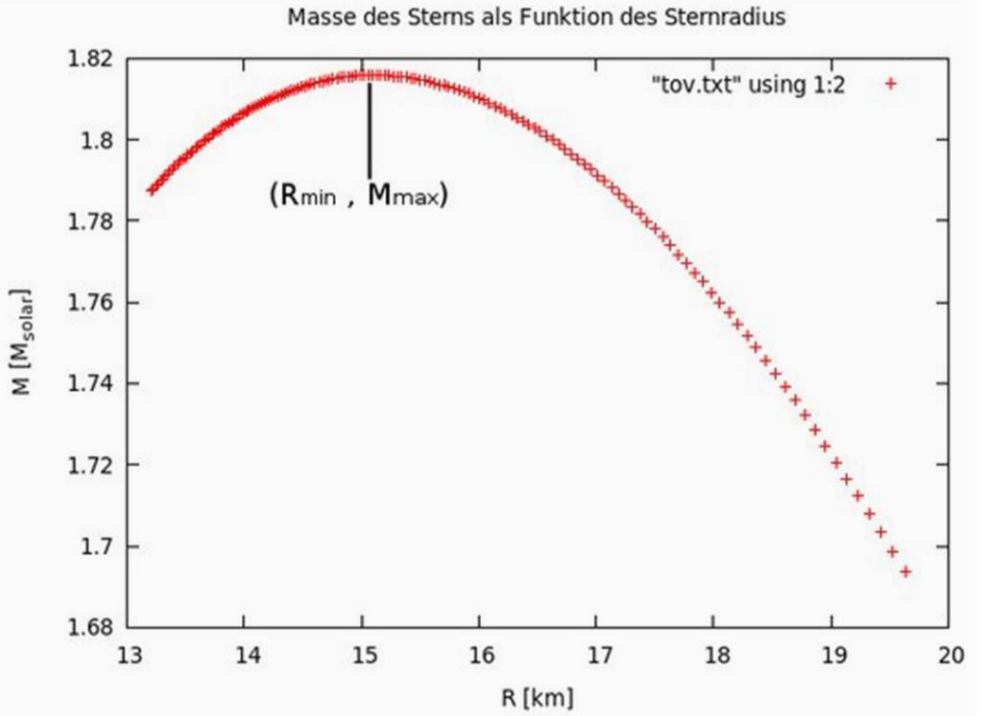
Radialer Wurf eines Probekörpers in der Nähe eines nichtrotierenden schwarzen Lochs

Berechnung eines Neutronensterns

Berechnung eines Weißen Zwergs

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.

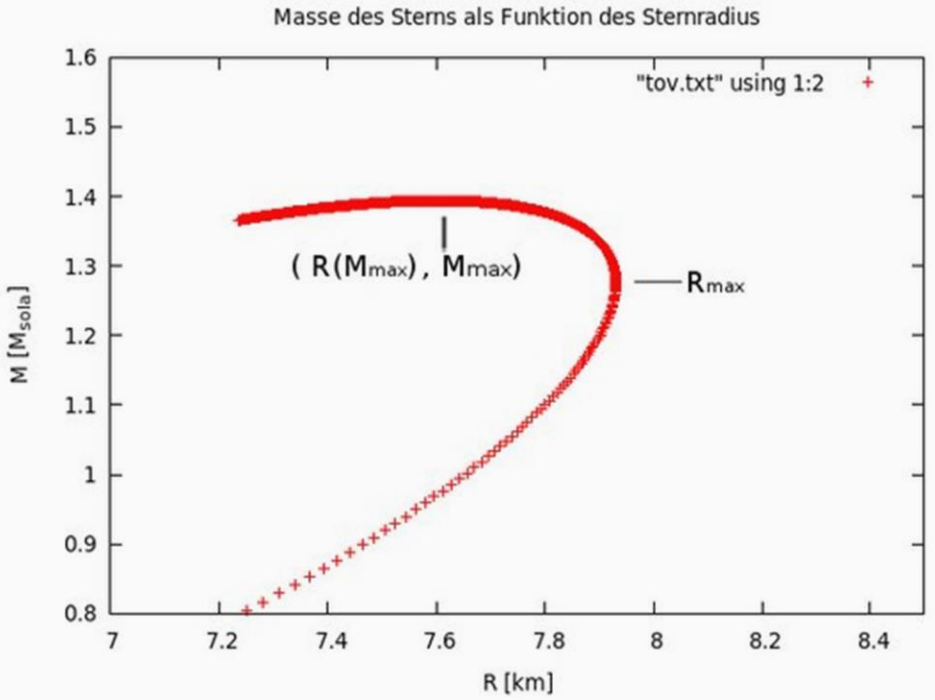


M_{max} = [ ] , R_{min} = [ ]

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.

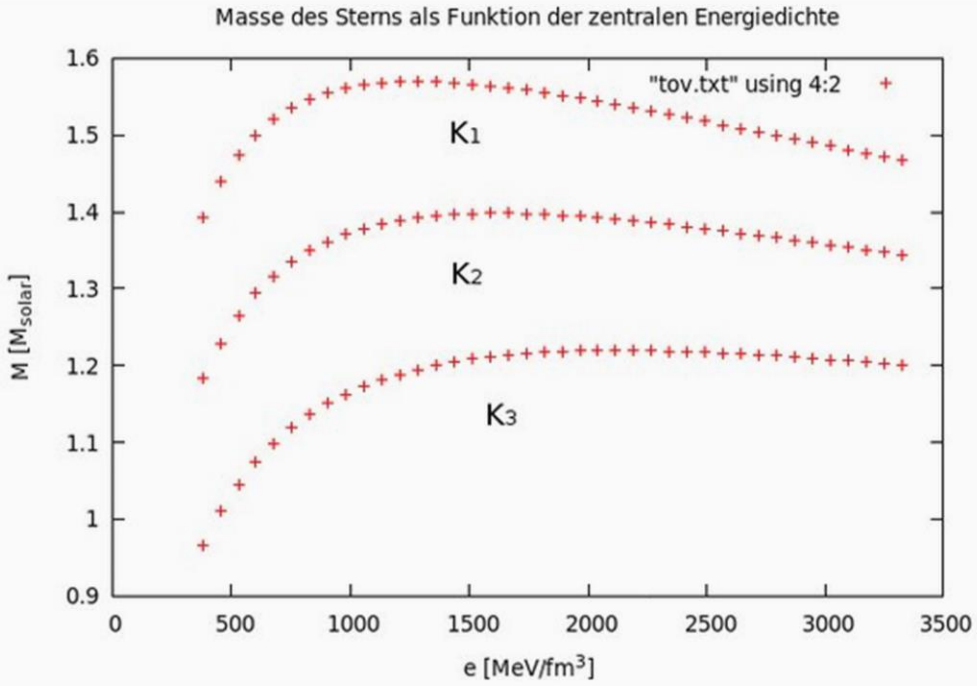


M_{max} = , $R(M_{max})$ = , R_{max} =

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 $M_{max} = 1.735147\ [\mathrm{M}_{\odot}]$.



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

Antwort einreichen Versuche 0/20