Allgemeine Relativitätstheorie mit dem Computer

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

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

MATTHIAS HANAUSKE

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

7. Vorlesung

Numerisches Lösen der TOV Gleichungen

mittels des Eulerverfahrens

Allgemeine Relativiätstheorie mit dem Computer: Teil II

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

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

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

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

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

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

TOV-Gleichungen: Numerisches Vorgehen

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

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

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

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

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

TOV-Gleichungen: Numerisches Vorgehen

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

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

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

C++ Lösen der TOV-Gleichung

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

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

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

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

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

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

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

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

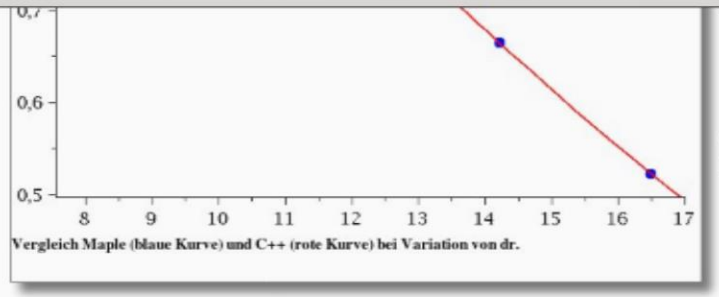
Deklaration der nötigen **Variablen** und der Zustandsgleichungsfunktion

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

TOV-Gleichungen

Ausgabe auf dem Bildschirm

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.

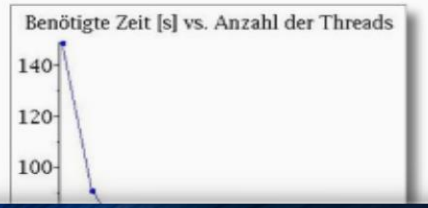
```
Datei  Bearbeiten  Ansicht  Lesezeichen  Einstellungen  Hilfe
hanauske@ITPRelAstro-Aspire-VN7-591G:~$ c++ -fopenmp TOV_parallel_omp.cpp
hanauske@ITPRelAstro-Aspire-VN7-591G:~$ ./a.out
35
Neutronensternradius [km]           = 9.13464
Neutronensternmasse [Sonnenmassen]  = 1.20785
00-Metrikkomponente im Sternzentrum = 0.198145
30
Neutronensternradius [km]           = 9.50537
Neutronensternmasse [Sonnenmassen]  = 1.21506
00-Metrikkomponente im Sternzentrum = 0.220643
25
Neutronensternradius [km]           = 9.94998
Neutronensternmasse [Sonnenmassen]  = 1.21927
00-Metrikkomponente im Sternzentrum = 0.248056
20
Neutronensternradius [km]           = 10.4976
Neutronensternmasse [Sonnenmassen]  = 1.21836
00-Metrikkomponente im Sternzentrum = 0.28223
15
Neutronensternradius [km]           = 11.1977
Neutronensternmasse [Sonnenmassen]  = 1.20841
00-Metrikkomponente im Sternzentrum = 0.326145
10
Neutronensternradius [km]           = 12.1444
```

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

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.

```
    //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

```
    //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
//Aufteilung der zu berechnenden Sterne auf die einzelnen MPI Prozesse
```

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

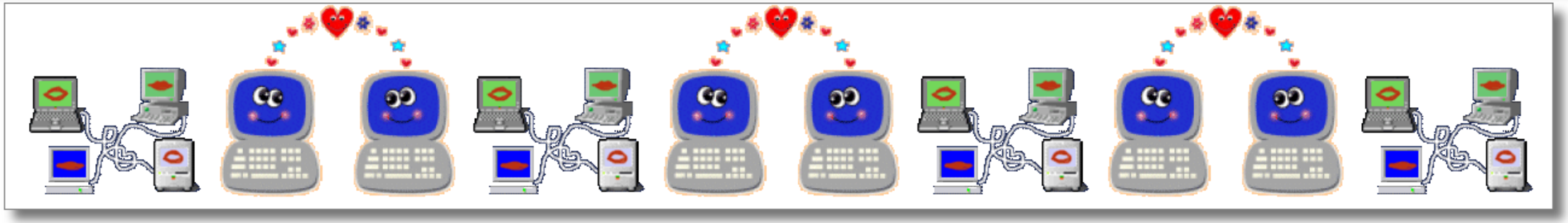
Eine sinnvolle Aufteilung der einzelnen Aufgaben des TOV-Programms auf die jeweiligen Prozesse kann man z.B. realisieren, indem die 150 (anz=150) zu berechnenden Neutronensterne auf die jeweiligen Prozesse aufteilt werden. In dem die for-Schleife von einem prozessabhängigen Startwert anfängt und in Schrittweiten '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

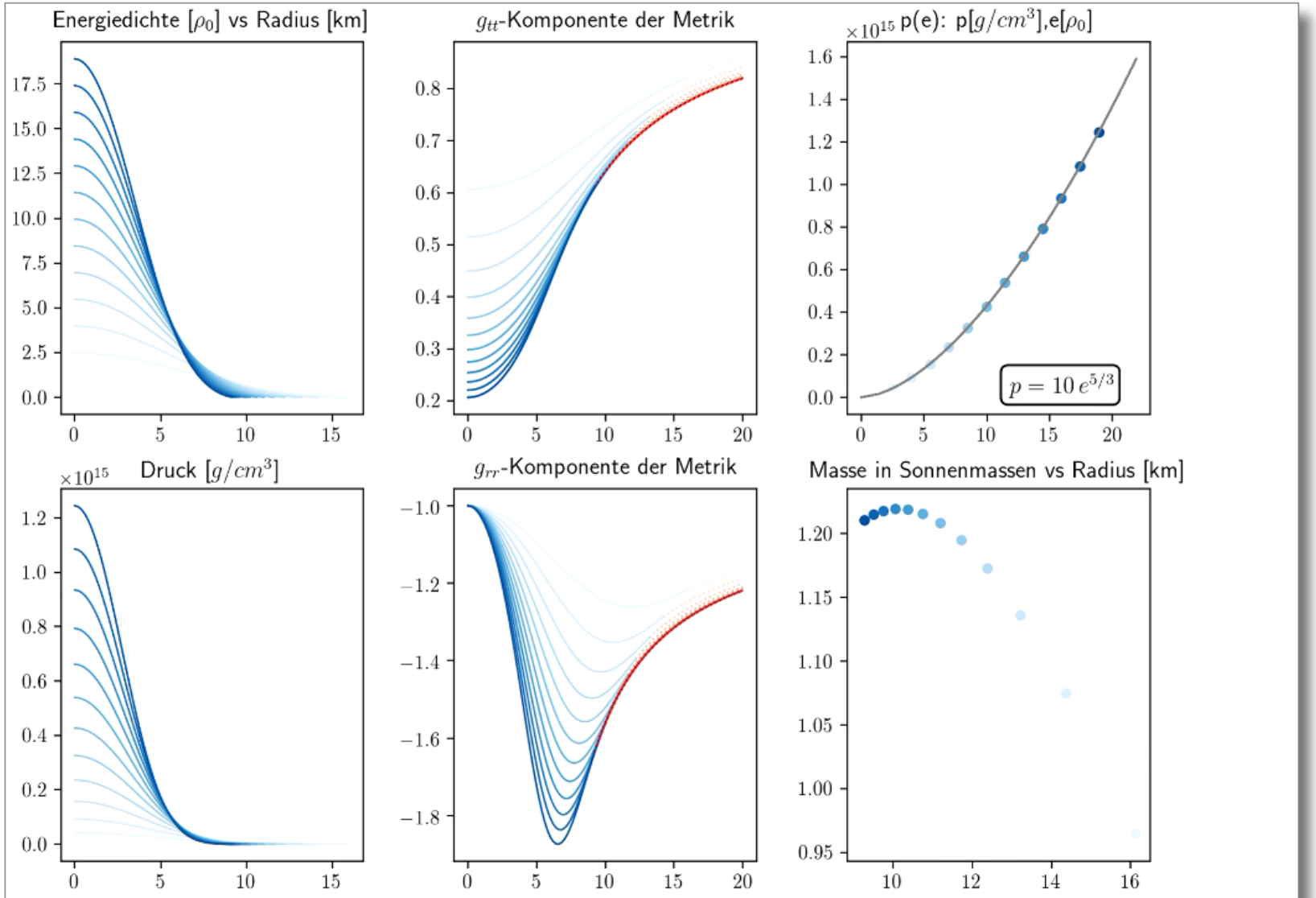
```
  //Geordnete Ausgabe der Masse, des Radius, der zentralen g00-Metrikkomponente und der zentralen Energiedichte in die Ausgabedatei
  //Die Ausgabe erfolgt nur von dem Prozess 0, der zunaechst alle berechneten und an ihn gesendeten Daten empfaengt
  if (id==0)
  {
    for (int proc=1;proc<psize;proc++)
    {
      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.

Das Python- Programm



Berechnung einer Sequenz von Neutronensternen mittels eines Python Skripts (siehe TOV-Sequence-plot2.py).

Berechnung einer Sequenz von Neutronensternen mittels Python (ohne Parallelisierung)

Dieser Unterpunkt des zweiten Teils der Vorlesung gibt eine Einführung in die Programmierung mit Python. Ausgehend von der, im ersten Teil hergeleiteten Tollmann-Openheimer-Volkov Gleichung, wird mittels des einfachen Euler-Verfahrens die Differentialgleichung in Python implementiert (siehe TOV-Sequence-simple.py, entspricht der C++ Version 'TOV sequentielle Version 1' aus Teil 2.1). Die numerisch berechneten Werte des Neutronensternradius und seiner gravitativen Masse werden am Ende des Programs im Terminal ausgegeben.

Im nächsten Schritt wird eine Eigenschaft der berechneten Neutronensterne in einem Diagramm ausgegeben (siehe TOV-Sequence-plot.py) und danach mehrere Diagramme (siehe TOV-Sequence-plot1.py). Die nebenstehende Abbildung zeigt die Ergebnisse des Python Skripts TOV-Sequence-plot2.py in einer Animation. Hierbei wurden die einzelnen sequentiell berechneten Sterne als Bilder gespeichert und mittels 'ffmpeg -framerate 1 -i './img-%03d.jpg' -c:v libx264 -vf "scale=trunc(iw/2)*2:trunc(ih/2)*2" -s 2000x2000 tovpython.mp4' in einem Film zusammengefügt.

Das Python- Programm (simple)

```
import math

# Definition der Zustandsgleichung
def eos(p):
    e=math.pow(p/10.0,3.0/5);
    return e;

# Variableninitialisierung
dr=0.000001;
dec=0.0001;

# for Schleife zur Berechnung mehrerer Sterne
for i in range(0,5):
    M=0;
    r=math.pow(10,-14);
    p=10*math.pow(0.0005+i*dec,5.0/3);
    nu=0;

    # while Schleife (Numerische Loesung der TOV-Gleichung)
    while p > 0:
        e=eos(p);                                          #Wert der Energiedichte bei momentanen Druck
        dM=4*math.pi*e*r*r*dr;                             #Massenzunahme bei momentanem r und Schrittweite dr
        dp=-(p+e)*(M+4*math.pi*r*r*r*p)/(r*(r-2*M))*dr;    #Druckzunahme bei momentanem r und Schrittweite dr (TOV-Gleichung)
        dnu=(M + 4*math.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

    nu=math.log(1-2*M/r)/2-nu;

    # Ausgabe der Masse und des Radius (im Terminal)
    print("i = {}".format(i))
    print("Neutronensternradius [km]           = {}".format(r))
    print("Neutronensternmasse [Sonnenmassen]  = {}".format(M/1.4766))
    print("00-Metrikkomponente im Sternzentrum = {}".format(math.exp(2*nu)))
```

Das Python- Programm (mit plot)

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

# Definition der Zustandsgleichung
def eos(p):
    e=math.pow(p/10.0,3.0/5)
    return e

# Variableninitialisierung
dr=0.00001
dec=0.0001

numpoints=100
plot_rmax=20
plot_dr=plot_rmax/np.float(numpoints)
plotstar=np.empty([numpoints,5])

numstars=5
cmap = plt.cm.Blues
line_colors = cmap(np.linspace(0,1,numstars+4))

# for Schleife zur Berechnung mehrerer Sterne
for i in range(0,numstars):
    M=0
    r=math.pow(10,-14)
    p=10*math.pow(0.0005+i*dec,5.0/3)
    nu=0
    plot_r=0
    plot_i=0

    # while Schleife (Numerische Loesung der TOV-Gleichung)
    while p > 0:
        e=eos(p)                                          #Wert der Energiedichte bei momentanen Druck
        dM=4*math.pi*e*r*r*dr                             #Massenzunahme bei momentanem r und Schrittweite dr
        dp=-(p+e)*(M+4*math.pi*r*r*r*p)/(r*(r-2*M))*dr    #Druckzunahme bei momentanem r und Schrittweite dr (TOV-Gleichung)
        dnu=(M + 4*math.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
        if r > plot_r:
            plotstar[plot_i].flat[0] = r
            plotstar[plot_i].flat[1] = e
            plotstar[plot_i].flat[2] = p
            plotstar[plot_i].flat[3] = M
            plotstar[plot_i].flat[4] = nu
            plot_r=plot_r+plot_dr
            plot_i=plot_i + 1
```

Das Python- Programm (mit plot)

```
    nu=math.log(1-2*M/r)/2-nu

    #Ausgabe der Masse und des Radius (im Terminal)
    print("i = {}".format(i))
    print("Neutronensternradius [km]           = {}".format(r))
    print("Neutronensternmasse [Sonnenmassen]  = {}".format(M/1.4766))
    print("00-Metrikkomponente im Sternzentrum = {}".format(math.exp(2*nu)))

    # Plotten der Energiedichte, des Druckes oder der Masse als Funktion des Radius
    plt.plot(plotstar[0:plot_i,0],plotstar[0:plot_i,1],c=line_colors[i+2], linewidth=1, linestyle='-')
#    plt.plot(plotstar[0:plot_i,0],plotstar[0:plot_i,2],c=line_colors[i+2], linewidth=1, linestyle='-')
#    plt.plot(plotstar[0:plot_i,0],plotstar[0:plot_i,3],c=line_colors[i+2], linewidth=1, linestyle='-')
plt.show()
```

Das Python- Programm (mit 6 Plots)

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

# Definition der Zustandsgleichung
def eos(p):
    e=math.pow(p/10.0,3.0/5)
    return e

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

# Gitter zum Plotten von sechs unterschiedliche Diagramme definieren
plt.figure(0)
gs = gridspec.GridSpec(2, 3, width_ratios=[1,1,1], wspace=0.3)
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)

#Konstanten
normalnuc=0.000201054535 # normale nukleare Kerdichte (Wert/Umrechnungsfaktor)
normalnuc1=2.705*math.pow(10,14) # normale nukleare Kerdichte (g/cm3)

# Variableninitialisierung
energy0=0.0005
dr=0.00001
dec=0.0003
numstars=12
```

Das Python- Programm (mit 6 Plots)

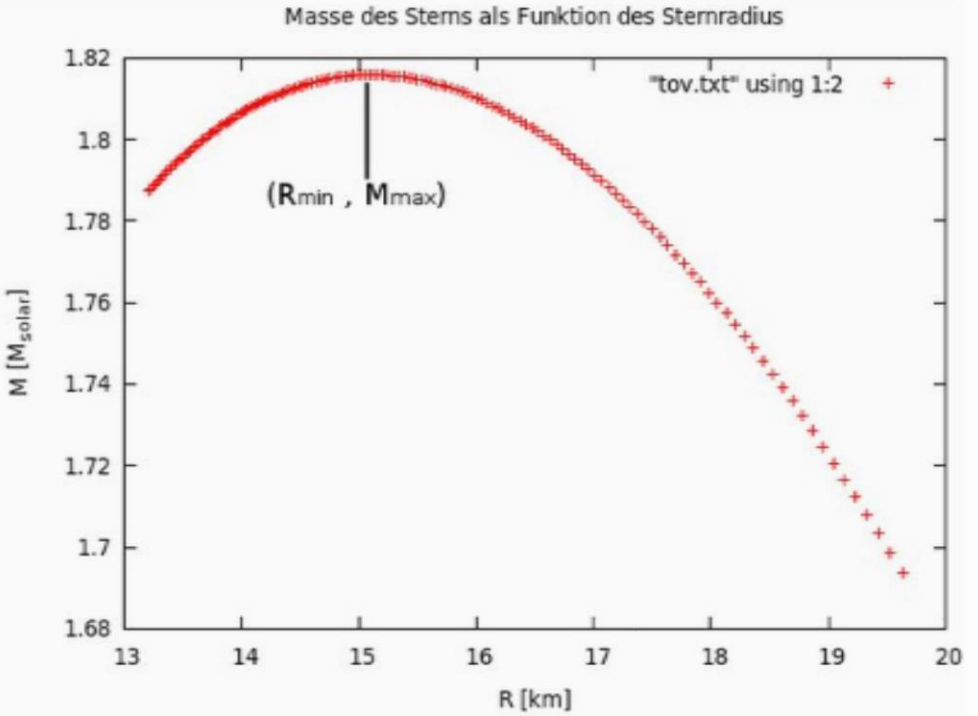
```
    # Plotten der einzelnen Teildiagramme
    ccolor=line_colors[i+2]
    ax1.plot(plotstar[0:plot_i,0],plotstar[0:plot_i,1]/normalnuc,c=line_colors[i+2], linewidth=1, linestyle='-')
    ax2.plot(plotstar[0:plot_i,0],plotstar[0:plot_i,5],c=line_colors[i+2], linewidth=1, linestyle='-')
    ax2.plot(np.linspace(plotstar[plot_i-1,0],20,30),(1-2*M/np.linspace(plotstar[plot_i-1,0],20,30)),c=line_colors1[i+2], linewidth=1, linestyle=':')
    ax2.plot(plotstar[0:plot_i,0],plotstar[0:plot_i,5],c=line_colors[i+2], linewidth=1, linestyle='-')
#     ax3.plot(plotstar[0:plot_i,0],plotstar[0:plot_i,3]/1.4766,c=line_colors[i+2], linewidth=1, linestyle='-')
    ax3.plot(eos_e,eos_p,c='Grey', linewidth=1, linestyle='-')
    ax3.scatter(plotstar[0,1]/normalnuc,plotstar[0,2]/normalnuc*normalnuc1,s=20,c=[line_colors[i+2]],marker="o")
    ax4.plot(plotstar[0:plot_i,0],plotstar[0:plot_i,2]/normalnuc*normalnuc1,c=line_colors[i+2], linewidth=1, linestyle='-')
    ax5.plot(plotstar[0:plot_i,0],plotstar[0:plot_i,6],c=line_colors[i+2], linewidth=1, linestyle='-')
    ax5.plot(np.linspace(plotstar[plot_i-1,0],20,30),-1/(1-2*M/np.linspace(plotstar[plot_i-1,0],20,30)),c=line_colors1[i+2], linewidth=1, linestyle=':')
    ax6.scatter(r,M/1.4766,s=20,c=[line_colors[i+2]],marker="o")

    # Plotten der Zustandsgleichung in das Bild
    textstr1=r'$p=10\,e^{5/3}$'
    ax3.text(20, 0, textstr1, fontsize=12, verticalalignment='bottom', horizontalalignment='right', bbox=props)

    # Plotten der einzelnen Titel der Teildiagramme
    ax1.set_title(r'Energiedichte [$ \rho_{0}$] vs Radius [km]')
    ax2.set_title(r'$g_{tt}$-Komponente der Metrik')
#     ax3.set_title('Masse m(r) in Sonnenmassen')
    ax3.set_title(r'p(e): p[${g}/{cm^{3}}$],e[$ \rho_{0}$]')
    ax4.set_title(r'Druck [${g}/{cm^{3}}$]')
    ax5.set_title(r'$g_{rr}$-Komponente der Metrik')
    ax6.set_title('Masse in Sonnenmassen vs Radius [km]')
plt.show()
```

Das Python- Programm (mit 6 Plots als Bild gespeichert)

```
    # Plotten der einzelnen Teildiagramme
    ccolor=line_colors[i+2]
    ax1.plot(plotstar[0:plot_i,0],plotstar[0:plot_i,1]/normalnuc,c=line_colors[i+2], linewidth=1, linestyle='-')
    ax2.plot(plotstar[0:plot_i,0],plotstar[0:plot_i,5],c=line_colors[i+2], linewidth=1, linestyle='-')
    ax2.plot(np.linspace(plotstar[plot_i-1,0],20,30),(1-2*M/np.linspace(plotstar[plot_i-1,0],20,30)),c=line_colors1[i+2], linewidth=1, linestyle=':')
    ax2.plot(plotstar[0:plot_i,0],plotstar[0:plot_i,5],c=line_colors[i+2], linewidth=1, linestyle='-')
#     ax3.plot(plotstar[0:plot_i,0],plotstar[0:plot_i,3]/1.4766,c=line_colors[i+2], linewidth=1, linestyle='-')
    ax3.plot(eos_e,eos_p,c='Grey', linewidth=1, linestyle='-')
    ax3.scatter(plotstar[0,1]/normalnuc,plotstar[0,2]/normalnuc*normalnuc1,s=20,c=[line_colors[i+2]],marker="o")
    ax4.plot(plotstar[0:plot_i,0],plotstar[0:plot_i,2]/normalnuc*normalnuc1,c=line_colors[i+2], linewidth=1, linestyle='-')
    ax5.plot(plotstar[0:plot_i,0],plotstar[0:plot_i,6],c=line_colors[i+2], linewidth=1, linestyle='-')
    ax5.plot(np.linspace(plotstar[plot_i-1,0],20,30),-1/(1-2*M/np.linspace(plotstar[plot_i-1,0],20,30)),c=line_colors1[i+2], linewidth=1, linestyle=':')
    ax6.scatter(r,M/1.4766,s=20,c=[line_colors[i+2]],marker="o")

    # Plotten der Zustandsgleichung in das Bild
    textstr1=r'$p=10\,e^{5/3}$'
    ax3.text(20, 0, textstr1, fontsize=12, verticalalignment='bottom', horizontalalignment='right', bbox=props)

    # Plotten der einzelnen Titel der Teildiagramme
    ax1.set_title(r'Energiedichte [$ \rho_{0}$] vs Radius [km]')
    ax2.set_title(r'$g_{tt}$-Komponente der Metrik')
#     ax3.set_title('Masse m(r) in Sonnenmassen')
    ax3.set_title(r'p(e): p[${g}/{cm^{3}}$],e[$ \rho_{0}$]')
    ax4.set_title(r'Druck [${g}/{cm^{3}}$]')
    ax5.set_title(r'$g_{rr}$-Komponente der Metrik')
    ax6.set_title('Masse in Sonnenmassen vs Radius [km]')

    # Ausgabe einer Sequenz von Bildern als jpg oder pdf
    saveFig="./output/img-"+"{:0>3d}".format(i)+".jpg"
    plt.savefig(saveFig, dpi=200,bbox_inches="tight",pad_inches=0.05,format="jpg")
#   saveFig="./output/img-"+"{:0>3d}".format(i)+".pdf"
#   plt.savefig(saveFig,bbox_inches="tight",pad_inches=0.05,format="pdf")
plt.show()
```

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

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

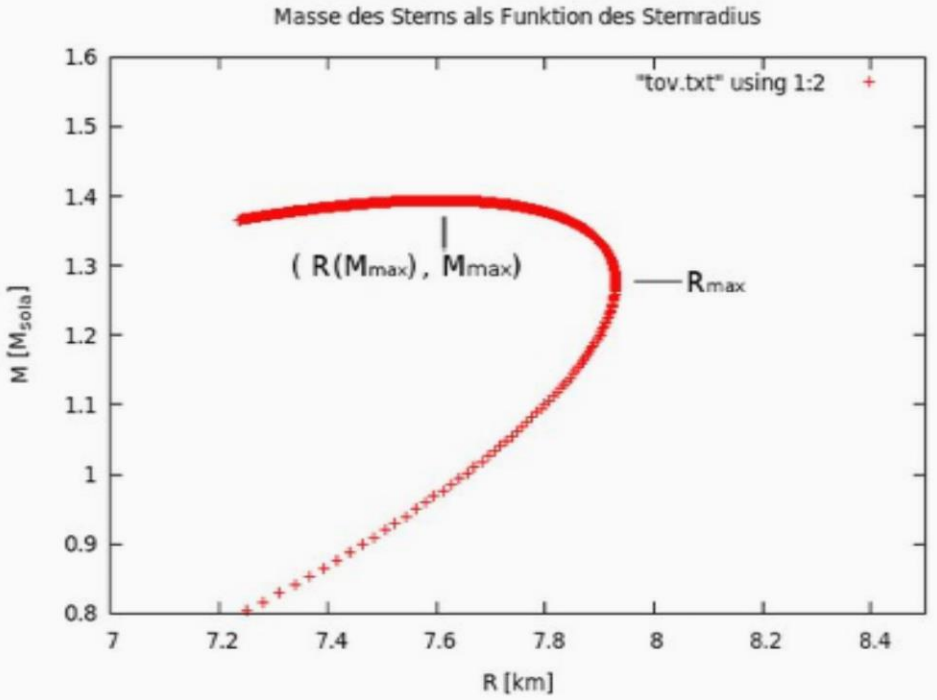


M_{max} = , R_{min} =

Antwort einreichen Versuche 0/20

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

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

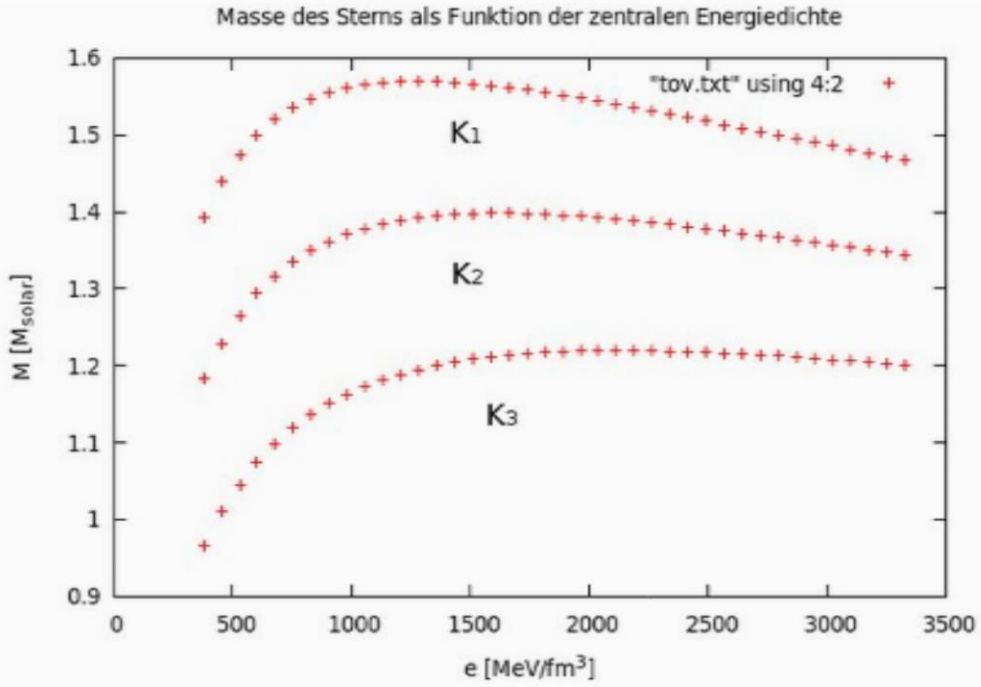


M_{max} = , $R(M_{max})$ = , R_{max} =

Antwort einreichen Versuche 0/20

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

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



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

Antwort einreichen Versuche 0/20