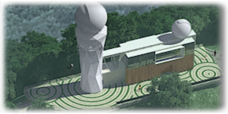
Spectral methods in Lorene

Yu Liu[1]

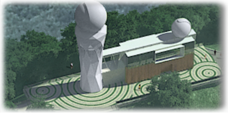
[1]Department of Astronomy
Huazhong University of Science and Technology
yul@hust.edu.cn

September 24, 2020

Table of Contents



Introduction



All numerical techniques is to approximate any function by polynomials, those being the only functions than a computer can exactly calculate. So a function u will be approximate by

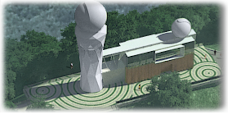
$$\hat{u} = \sum_{n=0}^{N} \hat{u}_n \Phi_n \tag{1.1}$$

where the Φ_n are polynomials and called the trial functions. **Depending on the choice of trial functions, one can generate various classes of numerical techniques.** [Grandclement, 2006]

Two types of numerical methods:

- **Spectral methods**: high order polynomials on a single domain.
- **Finite elements**: low order polynomials on many domains.

Orthogonal projection



Let us consider an interval $\Lambda = [x_{\rm min}, x_{\rm max}]$. In order to talk about basis, one needs to define a scalar product on Λ. If w is a positive function on Λ, one can define the scalar product of two functions f and g, with respect to the measure w as being

$$(f, g)_w = \int_\Lambda f(x)g(x)w(x)\mathrm{d}x. \tag{1.2}$$

Using this scalar product, one can find a set of orthogonal polynomials p_n, each of them of degree n. The set composed of those polynomials, up to a given degree N is a basis of $\mathbb{P}_N$.

One can then hope to represent any function u on Λ by its projection on the polynomials p_n. Doing so, we define the projection of u simply by

$$P_N u = \sum_{n=0}^{N} \hat{u}_n p_n(x), \tag{1.3}$$

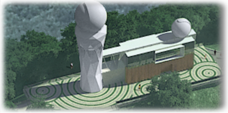
where the coefficients of the projections are given by $\hat{u}_n = \frac{(u,p_n)}{(p_n,p_n)}$. The difference between u and its projection is called the truncation error and one can show that it goes to zero when the order of the approximation increases:

$$\|u - P_n u\| \longrightarrow 0 \quad \text{when} \quad N \longrightarrow \infty. \tag{1.4}$$

Note

This seems very appealing but for the fact that one needs to calculate the $\hat{u}_n$ by computing integrals like $\int_\Lambda u(x)p_n(x)w(x)\mathrm{d}x$.

Gauss quadratures



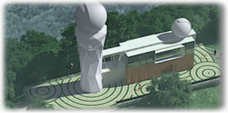
There exist $N+1$ positive reals w_n and $N+1$ reals x_n in Λ such that:

$$\forall f \in \mathbb{P}_{2N+\delta}, \quad \int_\Lambda f(x) w(x) \mathrm{d}x = \sum_{n=0}^{N} f(x_n) w_n. \tag{1.5}$$

The w_n are called the weights and the x_n **the collocation points.** The exact degree of applicability depends on the quadrature. The three usual choices are:

- Gauss: $\delta = 1$
- Gauss-Radau: $\delta = 0$ and $x_0 = x_{\min}$
- Gauss-Lobatto: $\delta = -1$ and $x_0 = x_{\min}$ and $x_N = x_{\max}$

Interpolation



If one applies the Gauss quadratures to approximate the coefficient of the expansion, one obtains

$$\tilde{u}_n = \frac{1}{\gamma_n} \sum_{j=0}^{N} u\left(x_j\right) p_n\left(x_j\right) w_j \quad \text{with} \quad \gamma_n = \sum_{j=0}^{N} p_n^2\left(x_j\right) w_j. \tag{1.6}$$

Let us precise that this is not exact in the sense that $\hat{u}_n \neq \tilde{u}_n$. However, the computation of $\hat{u}$ only requires to evaluate u at the $N+1$ collocation points.
The interpolant of u is then defined as the following polynomial

$$I_N u = \sum_{n=0}^{N} \tilde{u}_n p_n(x). \tag{1.7}$$

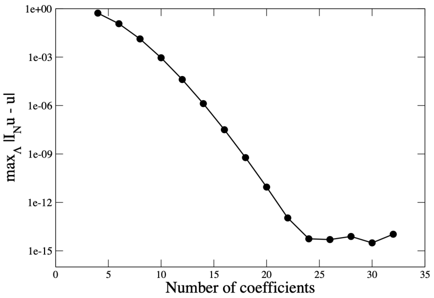
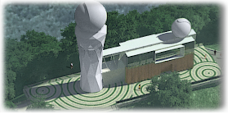


Figure: Maximum difference between $I_N u$ and u as a function of the degree of the approximation N.

We can observe the very general feature of spectral methods that **the error decreases exponentially**, until one reaches the machine accuracy (here 10^{-14}).
This very fast convergence explains why spectral methods are so efficient, especially compared to finite difference ones, where the error follows only a power-law in terms of N.

The derivative of the interpolant



One can simply approximate u' by the derivative of the interpolant:

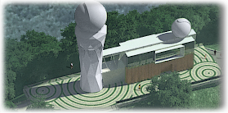
$$u'(x) \approx [I_N u]' = \sum_{n=0}^{N} \tilde{u}_n p_n'(x) \tag{1.8}$$

Such an approximation only requires the knowledge of the coefficients of u and how the basis polynomials are derived.

Note

The interpolation and the derivation are two operations that do not commute: $(I_N u)' \neq I_N(u')$.

Chebychev polynomials



The Chebyshev polynomials T_n are an orthogonal set on $[-1, 1]$ for the measure $w = \frac{1}{\sqrt{1-x^2}}$. More precisely one has

$$\int_{-1}^{1} \frac{T_n T_m}{\sqrt{1-x^2}} \mathrm{d}x = \frac{\pi}{2} (1 + \delta_{0n}) \delta_{mn}. \tag{1.9}$$

Chebyshev polynomials can be computed by knowing that $T_0 = 1$, $T_1 = x$ and by making use of the recurrence:

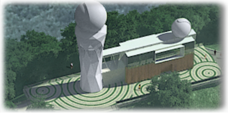
$$T_{n+1}(x) = 2xT_n(x) - T_{n-1}(x). \tag{1.10}$$

The derivatives of Chebychev polynomial

$$T_n'(x) = 2nT_{n-1}(x) + \frac{n}{n-2} T_{n-2}'(x), \quad n > 2 \tag{1.11}$$

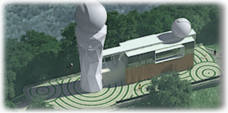
as well as $T_2'(x) = 4T_1(x)$, $T_1'(x) = T_0$ and, evidently, $T_0'(x) = 0$.

The weights and collocation points



The weights and collocation points associated with Chebyshev polynomials can be computed:

- Chebyshev-Gauss: $x_i = \cos\frac{(2i+1)\pi}{2N+2}$ and $w_i = \frac{\pi}{N+1}$
- Chebyshev-Gauss-Radau: $x_i = \cos\frac{2\pi i}{2N+1}$. The weights are $w_0 = \frac{\pi}{2N+1}$ and $w_i = \frac{2\pi}{2N+1}$
- Chebyshev-Gauss-Lobatto: $x_i = \cos\frac{\pi i}{N}$. The weights are $w_0 = w_N = \frac{\pi}{2N}$ and $w_i = \frac{\pi}{N}$



Type of problems

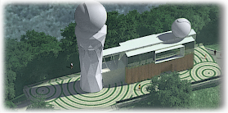
Let us consider a differential equation of the form

$$Lu(x) = S(x), \quad x \in [-1, 1] \tag{1.12}$$

where L is a linear differential operators.
The action of L on u can be expressed by a matrix L_{ij}. If the coefficients of u with respect to a given basis are the $\tilde{u}_i$ then

$$L(\sum_{k=0}^{N} \tilde{u}_k T_k(x)) \sim S(x)$$

Type of problems



Let us consider a differential equation of the form

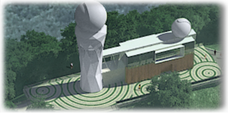
$$Lu(x) = S(x), \quad x \in [-1, 1] \tag{1.12}$$

where L is a linear differential operators.
The action of L on u can be expressed by a matrix L_{ij}. If the coefficients of u with respect to a given basis are the $\tilde{u}_i$ then

$$L(\sum_{k=0}^{N} \tilde{u}_k T_k(x)) \sim S(x)$$

$$\sum_{k=0}^{N} \tilde{u}_k \sum_{m=0}^{N} L_{mk} T_m(x) \sim S(x), \quad x \in [-1, 1] \tag{1.13}$$

Methods of weighted residuals

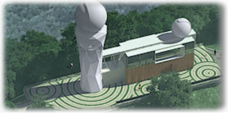


A function u is then an admissible solution of this system, if and only if
1) it fulfills the boundary conditions exactly (up to machine accuracy)
2) it makes the residual $R \equiv Lu - S$ small.
In order to quantify what this "small" means, the weighted residual method relies on $N + 1$ tests functions ξ_n and one asks that the scalar product of R with those functions is exactly zero:

$$(\xi_k, R) = 0, \quad \forall k \leq N \tag{1.14}$$

Depending on the choice of spectral basis and of test functions, one can generate various different types of spectral solvers.

The collocation method



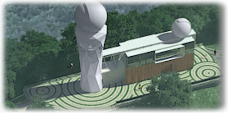
In the collocation method one uses continuous functions that are zero at all but one collocation point. They can be written as $\xi_i\left(x_j\right) = \delta_{ij}$. With such test functions, the residual equations are

$$\sum_{k=0}^{N} \tilde{u}_k \sum_{m=0}^{N} L_{mk} T_m(x_n) = S\left(x_n\right), \quad \forall n \leq N \tag{1.15}$$

the unknowns being the $\tilde{u}_k$. However, as such, this system does not admit a unique solution, due to the homogeneous solutions of L (i.e. the matrix associated with L is not invertible) and one has to impose boundary conditions.

In the collocation method, this is done by relaxing two equations (i.e. for $n = 0$ and $n = N$) and replacing them by the boundary conditions at $x = -1$ and $x = 1$.

General PDE solvers



$$H_i f_i = S_i \left(f_1, f_2 \ldots f_k\right) \quad \forall 0 \leq i < k \tag{1.16}$$

where H_i are differential operators (typically second order).

Iteration technique

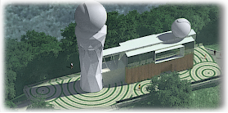
- Give an initial guess for the f_i;
- Computes the sources $S_i \left(f_1, f_2 \ldots . f_k\right)$;
- Invert the operators H_i;
- If the relative change in the f_i is small stop, else compute the new sources and loop.

We slow the change from step to step by using relaxation like:

$$f_i^{\mathrm{new}} = \lambda H_i^{-1}\left[S_i\right] + (1-\lambda) f_i^{\mathrm{old}}$$

where typical values $\lambda \approx 0.5$.

Multi-domain Spectral Method

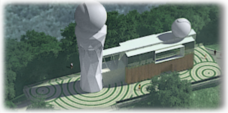


Spectral methods lose much of their accuracy when non-smooth functions are treated because of the so-called Gibbs phenomenon. The multi-domain spectral method circumvents the Gibbs phenomenon. The basic idea is to divide the space into domains chosen so that the physical discontinuities are located onto the boundaries between the domains. [Bonazzola et al., 1998]

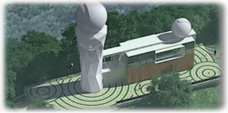
Examples

The simplest example is the case of a perfect fluid star, where two domains may be distinguished: the interior and the exterior of the star.

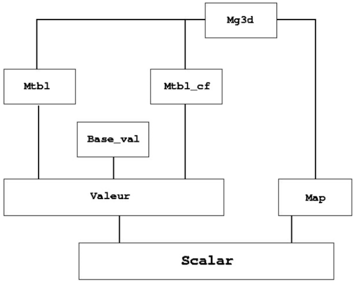
Table of Contents



Introduction



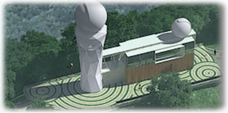
LORENE is a set of C++ classes. It provides tools to solve partial differential equations by means of multi-domain spectral methods.



- The class **Mg3D** stores Multi-domain grid of collocation points and takes into account symmetries;
- The class **Map** relates the numerical grid coordinates (ξ, θ', φ') to the physical ones (r, θ, φ);
- The class **Mtbl** stores values of a function on grid points;
- The class **Mtbl_cf** stores spectral coefficients of a function;
- The class **Base_val** contains information about the spectral bases;
- The class **Valeur** gathers a **Mtbl**, a **Mtbl_cf** and the **Base_val** to pass from one to the other.

Fields are represented using 3D spherical coordinates r, θ, φ and a spherical-like grid.

Header

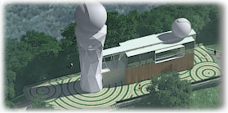


Any source file using Lorene should include the header

```
// C headers
#include <cstdlib>
#include <cassert>
#include <cmath>
// Lorene headers
#include "headcpp.h"     // standard input/output C++ headers (
    iostream, fstream)
#include "metric.h"      // classes Metric, Tensor, etc...
#include "nbr_spx.h"     // defines infinity as an ordinary
    number: __infinity
#include "graphique.h"   // for graphical outputs
#include "utilitaires.h"// utilities

using namespace Lorene ;

int main() {
  /* Here goes your code */
  return EXIT_SUCCESS ;
}
```



Mg3d

Multi-domain grid of collocation points on which the functions are evaluated to compute the spectral coefficients.

```
int nz = 3 ; // Number of domains
int nr = 7 ; // Number of collocation points in r
int nt = 5 ; // Number of collocation points in theta
int np = 8 ; // Number of collocation points in phi
```

Additional symmetries can be taken into account:

```
int symmetry_theta = SYM ; // symmetry with respect to the
    equatorial plane (z=0)
int symmetry_phi = NONSYM ; // invariance under the (x,y)->(-x
    ,-y) transform.
```

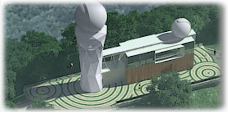
The last domain have a $1/r$ sampling.

```
bool compact = true ; // external domain is compactified
```

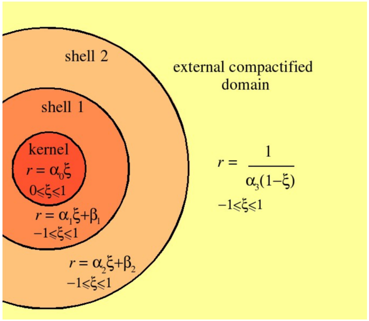
In each domain, the radial variable used is $\xi \in [-1, 1]$, or $\in [0, 1]$ for the nucleus.

```
// Multi-domain grid construction:
Mg3d mgrid(nz, nr, nt, np, symmetry_theta, symmetry_phi,
    compact) ;
```

Mappings

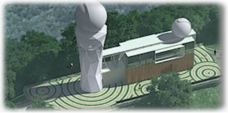


The boundary of each domain is chosen in order to coincide with a physical discontinuity.
A mapping relates, in each domain, the numerical grid coordinates (ξ, θ', φ') to the physical ones (r, θ, φ). The simplest class is **Map_af** for which the relation between and ξ and r is linear (nucleus + shells) or inverse (CED).


shell 2
external compactified domain
shell 1
kernel
$r = \alpha_0\xi$
$0 \leqslant \xi \leqslant 1$
$r = \dfrac{1}{\alpha_3(1-\xi)}$
$-1 \leqslant \xi \leqslant 1$
$r = \alpha_1\xi+\beta_1$
$-1 \leqslant \xi \leqslant 1$
$r = \alpha_2\xi+\beta_2$
$-1 \leqslant \xi \leqslant 1$

```
// radial boundaries of each domain:
double r_limits[] = {0., 1., 2., __infinity} ;
assert(nz==3) ; // since above array describes only 3 domains
// Construction of an affine mapping (Map_af)
Map_af map(mgrid, r_limits) ;
```

Scalar fields



The class Scalar gathers a Valeur and a mapping, it represents a scalar field defined on the spectral grid.

```cpp
// Various coordinates associated with the mapping
const Coord& r = map.r ;         // r field
const Coord& x = map.x ;         // x field
const Coord& y = map.y ;         // y field
// Setup of a regular scalar field
Scalar phi(map) ;
phi = x*exp(-r*r-y*y) ;
```

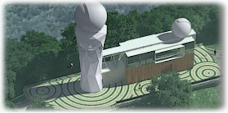
Accessors and modifier of values in a given domain

```cpp
// 0 at spatial infinity (instead of NaN !)
phi.set_outer_boundary(nz-1, 0) ;
```

Spectral base manipulation

```cpp
// Standar polynomial bases will be used to perform the
    spectral expansions
phi.std_spectral_base() ;
```

The dzpuis flag



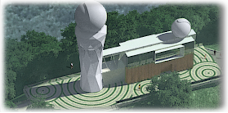
In the compactified external domain (CED), the variable $u = 1/r$ is used (up to a factor α). When computing the radial derivative (i.e. using the method **phi.dsdr()**) of a field f, one gets

$$\frac{\partial f}{\partial u} = -r^2 \frac{\partial f}{\partial r} \tag{2.1}$$

Use of an integer flag "dzpuis" for a scalar field f, which means that in the CED, one does not have f, but $r^{\mathrm{dzpuis}} f$ stored.

```
// Computation of the radial derivative
Scalar dphidr = phi.dsdr() ;
dphidr.dec_dzpuis(2) ;
```

Vector field



Lorene can handle a vector field V expressed in either of two types of components (i.e. using two orthonormal triads, of type **Base_vect**).

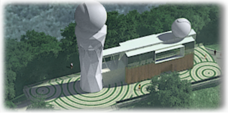
- the Cartesian triad $(e_x, e_y, e_z) = \left(\frac{\partial}{\partial x}, \frac{\partial}{\partial y}, \frac{\partial}{\partial z}\right)$
- the spherical triad $(e_r, e_\theta, e_\varphi) = \left(\frac{\partial}{\partial r}, \frac{1}{r}\frac{\partial}{\partial \theta}, \frac{1}{r\sin\theta}\frac{\partial}{\partial \varphi}\right)$

```
// Vector field defined by its cartesian components
Vector v_cart(map, CON, map.get_bvect_cart()) ;
// Change to spherical triad
v_cart.change_triad(map.get_bvect_spher()) ;
```

The covariance type of the indices is indicated by an integer which takes two values, defined in file **tensor.h**:

- COV: covariant index
- CON: contravariant index

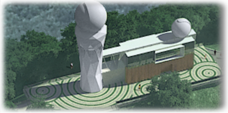
Metric



Components of the flat metric in an orthonormal cartesian frame

```
Sym_tensor g_uu(map, CON, map.get_bvect_cart()) ;
// write of a particular element (index i,j)
g_uu.set(1,1) = 1 ;
g_uu.set(2,2) = 1 ;
g_uu.set(3,3) = 1 ;
g_uu.set(1,2).annule_hard() ; // Sets it to zero in a hard way.
g_uu.set(1,3).annule_hard() ;
g_uu.set(2,3).annule_hard() ;
g_uu.std_spectral_base() ;
// 3-metric
Metric gam(g_uu) ;
```

Tensor calculus



- Tensorial product:

```
Tensor_sym tens3 = tens1 * tens2 ;
```

- Contraction:

```
// Contraction on two indices of a single tensor (trace).
Scalar scal = contract(tens, 0, 1) ; // 0 = first index, 1
    = second index, and so on...
// Contracting two tensors :
Tensor tens3 = contract(tens1, 1, tens2, 0) ;
```

- Raising an index with the metric **gam**:

```
Tensor tens = tens.up(1, gam) ;
```

- The covariant derivative of V with respect to the metric **gam**:

```
Tensor tens = v.derive_cov(gam) ;
```

- The Ricci tensor associated with the metric **gam**:

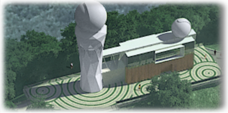
```
Sym_tensor ricci = gam.ricci() ;
```

- Lie derivative with respect to V:

```
Sym_tensor tens = tens.derive_lie(v) ;
```

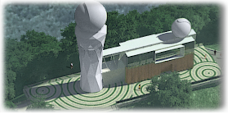
- and so on.

Numerical implementation



The spectral method amounts to reducing linear partial differential equations into a system of algebraic equations for the coefficients of the spectral expansions. The numerical code implementing the method is written in the LORENE.

Bin_star



Required executables and parfiles before run it.

- The parameter files for the code init_bin.C are: par_eos[1, 2].d par_grid[1, 2].d par_init.d

```
make init_bin
./init_bin
```

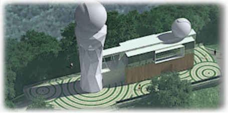
- There is only one parameter file for the code coal.C: parcoal.d

```
make coal
./coal
```

```
Bin_star
 ├─Makefile
 ├─init_bin.C
 ├─par_eos1.d
 ├─par_eos2.d
 ├─par_grid1.d
 ├─par_grid2.d
 ├─par_init.d
 ├─coal.C
 └─parcoal.d
```

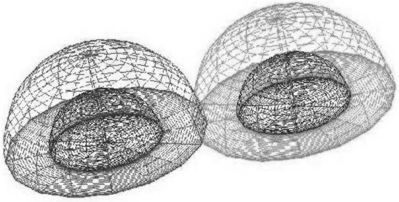
(under Lorene/Codes/)

./init_bin output is the unrelaxed binary configuration with given stellar models and specified binary separation. To generate a relaxed binary configuration, we need the **./coal**.



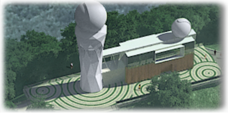
Grid

Two spherical coordinate systems are introduced, one centered on each star; this results in a precise description of the stellar interiors. The computational domain covers the whole space so that exact boundary conditions are set to infinity. [Gourgoulhon et al., 2001]



```
# Multi-grid parameters
######################
5 nz: total number of domains
1 nzet: number of domains inside the star
17   nt: number of points in theta (the same in each domain)
16   np: number of points in phi   (the same in each domain)
# Number of points in r and (initial) inner boundary of each domain:
33   0.  <-   nr   &   min(r)  in domain 0  (nucleus)
33   1.  <-   nr   &   min(r)  in domain 1
33   2.  <-   nr   &   min(r)  in domain 2
33   4.  <-   nr   &   min(r)  in domain 2
33   8.  <-   nr   &   min(r)  in domain 2
```

Equation of State



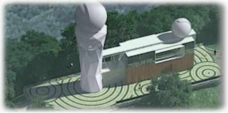
EoS data are to be stored in a formatted file **par_eos.d**. The fist line must start by the EOS number, see LORENE's Refguide.

- 1 = relativistic polytropic EOS (class Eos_poly)
- 17 = CompOSE (Tabulated EOS)
- 110 = Multi-polytropic EOS (class Eos_multi_poly)

The second line in the file should contain a name given by the user to the EOS. The following lines should contain the EOS parameters (one parameter per line), in the same order than in the class declaration.

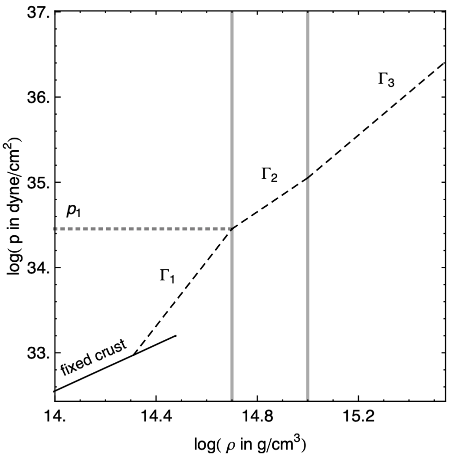
```
1 Type of the EOS
relativistic polytropic EOS
2.       adiabatic index gamma
0.0332   pressure coefficient kappa [rho_nuc c^2 / n_nuc^gamma]
1.       mean particle mass [m_b]2
```

Multi-polytropic EOS

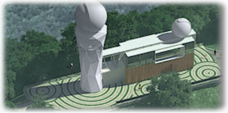


We used seven polytropic pieces, each corresponding to a different density interval. The four lower density pieces are the same for each EOS and come from the fitting of the crust. They represent, in increasing density order, a non-relativistic electron gas, a relativistic electron gas, the neutron drip regime, and the NS inner crust in the density interval between neutron drip and the nuclear saturation density. The three high density pieces, instead, are different for each NS core EOS model. [Read et al., 2009]

```
110      Type of the EOS
Multi-polytropic EOS
7        number of polytropes
1.58425 array of adiabatic index
1.28733
0.62223
1.35692
3.005
2.988
2.851
6.8011e-09 kappa
3.53623 logP1
7.3875  array of logRho
11.5779
12.4196
14.165
14.7
15
0.       array of percentage
0.
0.
0.
0.
0.
```



Tabulated Equation of state

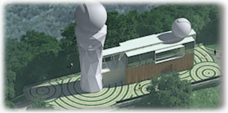


- Taken from the stellarcollapse database. (XXX.h5).
  Then use the python script called slicetable.py and scripts.py to get the equation of state in tabulated EOS format as read by LORENE. You will need to mention at the temperature at which you want to slice the table.

```
17   Type of the EOS
0    0: standard format
Tabulated EoS
/full/path/to/the/eos/table/name_of_the_table
```

- Taken from the CompOSE database (XXX.nb and XXX.thermo).

```
17   Type of the EOS
1    1: CompOSE format
Tabulated EoS
/full/path/to/the/eos/table/name_of_the_table
```



resu.d

The output result is contained in a binary file called **resu.d**, which is readable by appropriate routines in Einstein Toolkit.

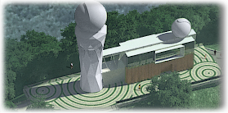
```
// Saveguard of the whole configuration
FILE* fresu = fopen("resu.d", "w") ;

star.get_mp().get_mg()->sauve(fresu) ;// writing of the grid
star.get_mp().sauve(fresu) ;           // writing of the mapping
star.get_eos().sauve(fresu) ;          // writing of the EOS
star.sauve(fresu) ;                    // writing of the star

fclose(fresu) ;
```

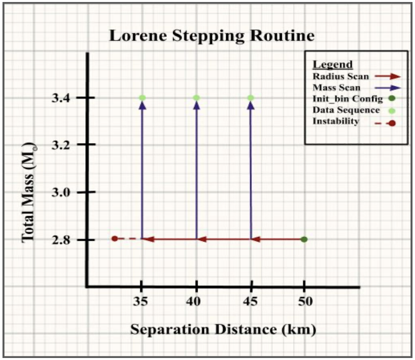
Check the outputs at the end, make sure there are no 'NaN'. Only then use the **resu.d** for evolving the binary.

Initial Data Library



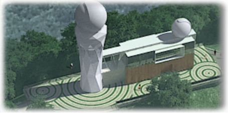
RIT Binary Neutron Star Initial Data Library

- Covers different piecewise polytropic approximants to physically motivated equations of state (Sly, AP3, AP4, WFF1, MPA1, MS1, MS1b)
- Covers different mass ratios (1, 1.14, 1.28, 1.428)
- Covers different separations starting from 50km and decreasing by 5km till 30km



The most stable results begin from low mass configurations, moving inwards to the desired radius, and then slowly increasing the target mass of the binary.

Einstein Toolkit



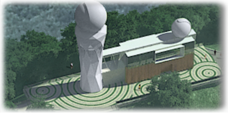
Einstein Toolkit contains three routines (under Cactus/arrangement/) that can read in publicly available data generated by the Lorene code. [Löffler et al., 2012]

- Meudon_Bin_BH: Binary black hole initial data (Lorene's **Bhole_binaire**);
- Meudon_Bin_NS: Binary neutron star initial data (Lorene's **Binaire**);
- Meudon_Mag_NS: Magnetized isolated neutron star initial data (Lorene's **Et_rot_mag**).

```
EinsteinInitialData
 |__Meudon_Bin_BH
 |__Meudon_Bin_NS
 |__Meudon_Mag_NS
```

Source codes contained in LORENE's Export subfolder

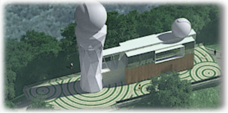
Meudon_Mag_NS



Code for reading a binary file containing data from a spectral computation of a rotating magnetized neutron stars with Lorene and exporting all the fields on a Cartesian grid.

```
// Reading of data
FILE* fich = fopen(filename, "r") ;
Mg3d spectral_grid(fich) ;
Map_et mapping(spectral_grid, fich) ;
Eos* p_eos = Eos::eos_from_file(fich) ;
Et_rot_mag star(mapping, *p_eos, fich) ; // For Meudon_Mag_NS
...
for (int i=0; i<np; i++) {
  double x0 = xx[i] * km ; // x, y, z in Lorene unit
  double y0 = yy[i] * km ;
  double z0 = zz[i] * km ;
  // polar coordinates centered on the star
  double r, theta, phi ;
  mapping.convert_absolute(x0, y0, z0, r, theta, phi) ;
  // Lapse function get from Et_rot_mag
  lapse[i] = sp_lapse.val_point(r, theta, phi) ;
} // End of loop on the points
```

Reference



Bonazzola, S., Gourgoulhon, E., and Marck, J.-A. (1998).
Numerical approach for high precision 3d relativistic star models.
PRD, 58(10):104020.

Gourgoulhon, E., Grandclément, P., Taniguchi, K., Marck, J.-A., and Bonazzola, S. (2001).
Quasiequilibrium sequences of synchronized and irrotational binary neutron stars in general relativity: Method and tests.
PRD, 63(6):064029.

Grandclement, P. (2006).
Introduction to spectral methods.
EAS Publications Series, 21:153–180.

Löffler, F., Faber, J., Bentivegna, E., Bode, T., Diener, P., Haas, R., Hinder, I., Mundim, B. C., Ott, C. D., Schnetter, E., Allen, G., Campanelli, M., and Laguna, P. (2012).
The einstein toolkit: a community computational infrastructure for relativistic astrophysics.
Classical and Quantum Gravity, 29(11):115001.

Read, J. S., Lackey, B. D., Owen, B. J., and Friedman, J. L. (2009).
Constraints on a phenomenologically parametrized neutron-star equation of state.
PRD, 79(12):124032.