Spectral methods in Lorene: regularity, symmetries, operators, ...

Jérôme Novak

Jerome.Novak@obspm.fr

Laboratoire de l'Univers et de ses Théories (LUTH)
CNRS / Observatoire de Paris, France

in collaboration with

Éric Gourgoulhon & Philippe Grandclément

November, 16 2005

Plan

Motivations and history

The numerical library Lorene (for Langage Objet pour la RElativité NumériquE) was initiated in 1997 by **Jean-Alain Marck** who, after realizing that the FORTRAN programming language the group has been using until then, was no longer adapted to the growing complexity of the numerical relativity codes.

Lorene is of course :

- a modular library written in C++,
- a collaborative effort (over 20 contributors),
- many users across the world,
- many published results in numerical relativity ...

thanks to :

- the `cvs` repository,
- fully-documented sources available on the web page `http ://www.lorene.obspm.fr`,
- a great effort to achieve portability across various systems / compilers.

Common features for many classes

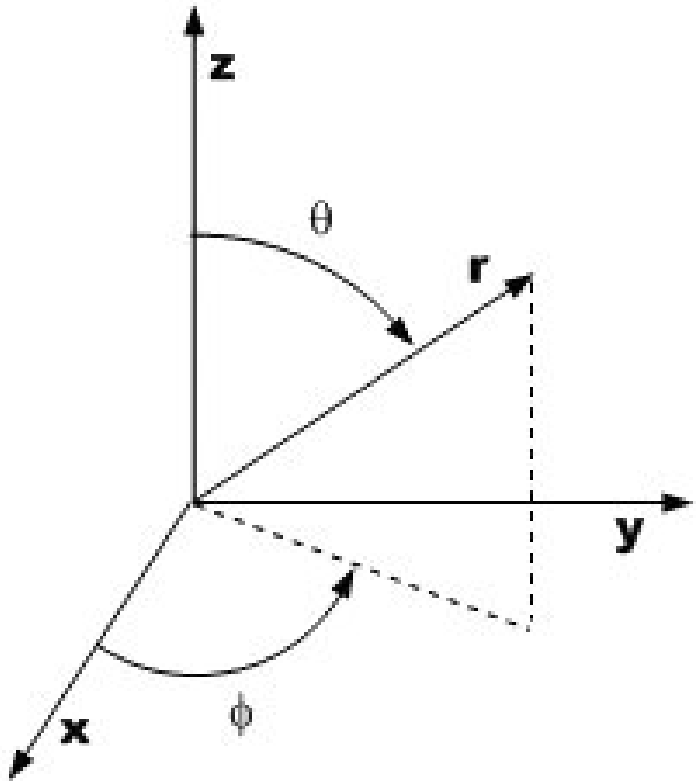
Most of classes (object types) in Lorene share some common functionalities :

- protected data, with readonly accessors often called `.get_XXX` and read/write accessors `.set_XXX`,
- an overload of the "<<" operator to display objects,
- a method for saving data into files and a constructor from a file,
- for container-like objects (arrays, fields...) a state (etat in French) flag indicating whether memory has been allocated :
  - `ETATQCQ` : ordinary state, memory allocated ⇒`set_etat_qcq()` ;
  - `ETATZERO` : null state, memory not allocated ⇒`set_etat_zero()` ;
  - `ETATNONDEF` : undefined state, memory not allocated ⇒`set_etat_nondef()` ;
  - + a method `annule_hard()` to fill with 0s ;
- external arithmetic operators (+, -, *, /) and mathematical functions (sin, exp, sqrt, abs, max, ...).

3D spherical polar coordinates

In almost all cases, fields are represented using 3D spherical coordinates r, θ, φ and a spherical-like grid :

- stars and black holes have spherical shapes,
- astrophysical systems are isolated : boundary conditions are defined for $r \to \infty$
- although spherical coordinates are singular (origin, z-axis), surfaces $r =$ constant are smooth.



$$\begin{aligned} x &= r \sin\theta \cos\varphi \\ y &= r \sin\theta \sin\varphi \\ z &= r \cos\theta, \end{aligned}$$

Regularity at the origin

Let $f(x, y, z)$ be an analytic function, it can be expanded near the origin in terms of Taylor series :

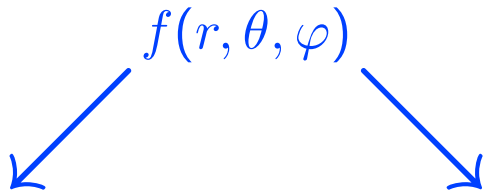
$$f(x, y, z) = \sum_{i=0}^{\infty} \sum_{j=0}^{\infty} \sum_{k=0}^{\infty} c_{ijk} x^i y^j z^k .$$

Changing the coordinates to spherical ones and after some amount of calculations and recasting $\cos\varphi$ and $\sin\varphi$ in $e^{i\varphi}$:

$$f(r, \theta, \varphi) = \sum_{\ell=0}^{\infty} \sum_{m=-\ell}^{\ell} r^\ell \sum_{i=0}^{\infty} a_{i\ell m} r^{2i} Y_\ell^m(\theta, \varphi).$$

Here $Y_\ell(\theta, \varphi) = P_\ell^m(\cos(\theta))\, e^{im\varphi}$ are the spherical harmonics with $P_\ell^m(\cos(\theta))$ being an associated Legendre polynomial in $\cos\theta$.

Spectral bases



ℓ EVEN

Radial base	θ base	φ base
Even Chebyshev	Even Fourier	Fourier
Even Chebyshev	Even Legendre	Fourier

ℓ ODD

Radial base	θ base	φ base
Odd Chebyshev	Odd Fourier	Fourier
Odd Chebyshev	Odd Legendre	Fourier

- Fourier series in θ $\Rightarrow$computation of derivatives or $1/\sin\theta$ operators ;
- associated Legendre polynomial in $\cos\theta$ $\Rightarrow$spherical harmonics $\Rightarrow$computation of the angular Laplace operator

$$\Delta_{\theta\varphi} \equiv \frac{\partial^2}{\partial\theta^2} + \frac{1}{\tan\theta}\frac{\partial}{\partial\theta} + \frac{1}{\sin^2\theta}\frac{\partial^2}{\partial\varphi^2}$$

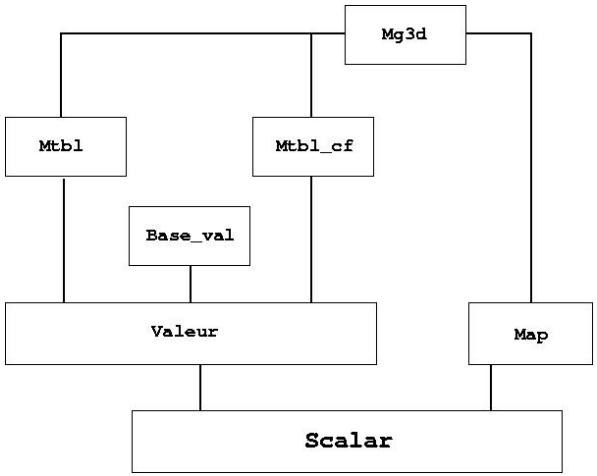
and inversion of the Laplace or d'Alembert operators.

Additional symmetries can be taken into account :

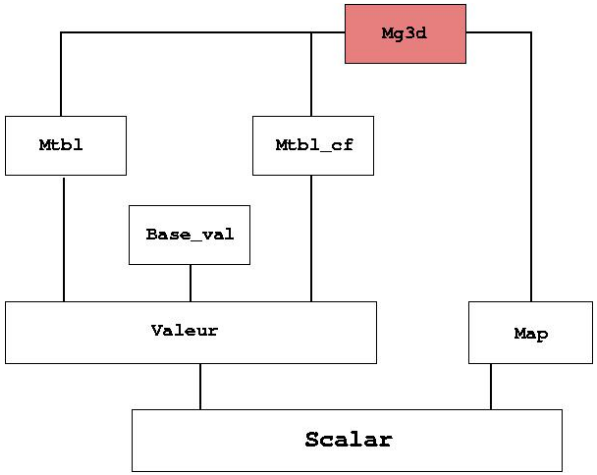
- the θ-*symmetry* : symmetry with respect to the equatorial plane ($z = 0$) ;
- the φ-*symmetry* : invariance under the $(x, y) \mapsto (-x, -y)$ transform.

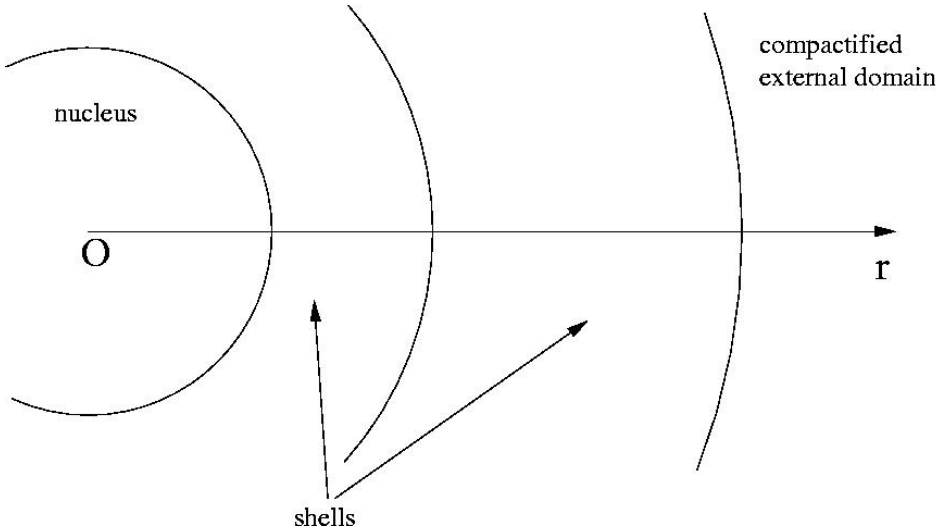
When required, only the angular functions which satisfy these symmetries are used for the decomposition and the grid is reduced in size.
The regularity condition on the z-axis is automatically taken into account by the spherical harmonics basis.

Spectral representation in Lorene



Mg3d

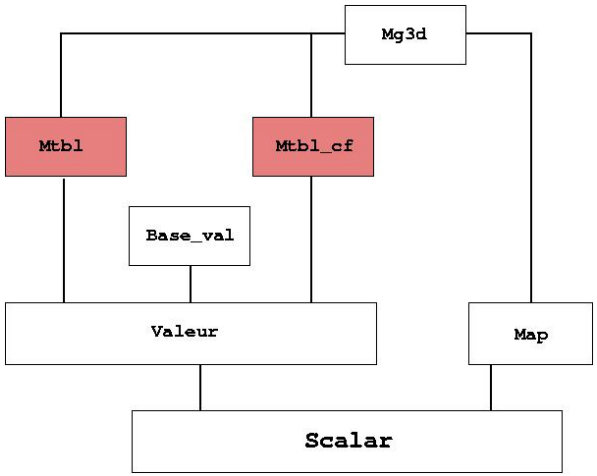




Multi-domain grid of collocation points on which the functions are evaluated to compute the spectral coefficients. It takes into account symmetries.

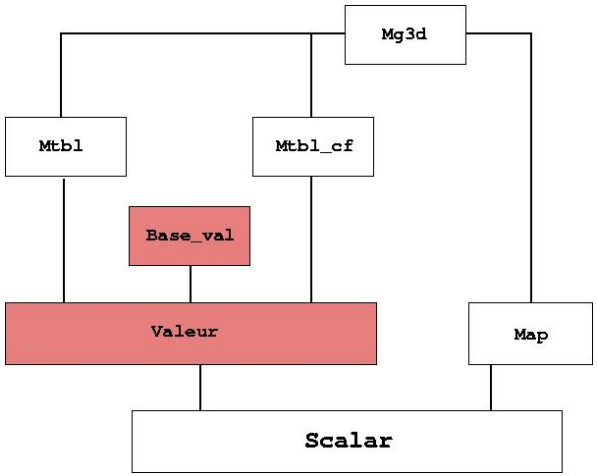
In each domain, the radial variable used is $\xi \in [-1, 1]$, or $\in [0, 1]$ for the nucleus.

Multigrid arrays



- The class `Mtbl` stores values of a function on grid points ; it depends on a multi-domain grid of type `Mg3d` and is merely a collection of 3D arrays `Tbl`.
- The class `Mtbl_cf` stores spectral coefficients of a function ; it has two more elements than the corresponding `Mtbl` in the φ-direction.

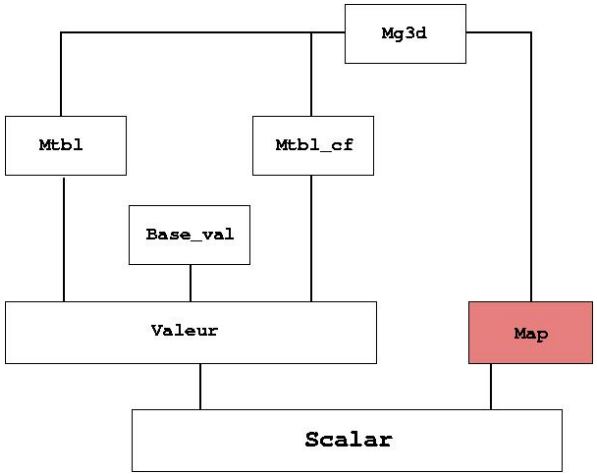
Base_val AND Valeur

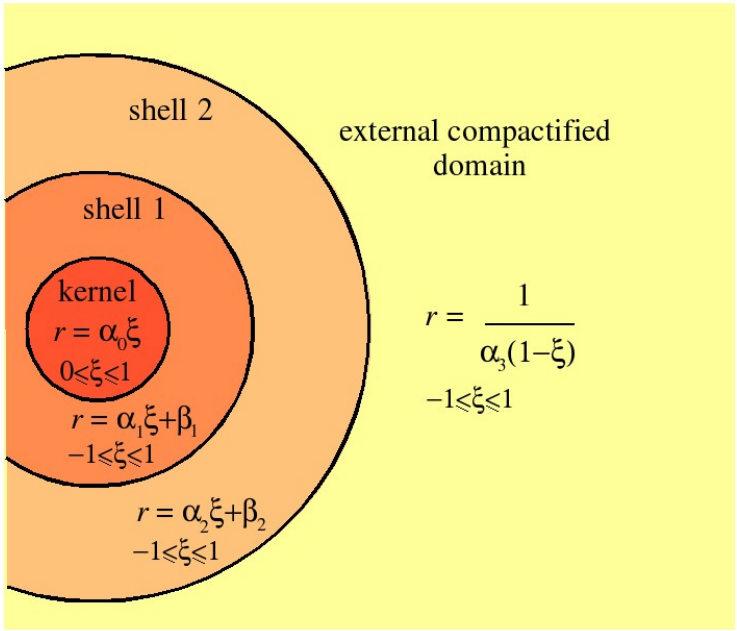


- The class `Base_val` contains information about the spectral bases used in each domain to transform from the function values on the grid points (`Mtbl`) to the spectral coefficients (`Mtbl_cf`).
- The class `Valeur` gathers a `Mtbl`, a `Mtbl_cf` and the `Base_val` to pass from one to the other.

An object of type `Valeur` can be initialized through its `Mtbl` (physical space) ; the coefficients can then be computed using the method `coef()` or `ylm()` for Fourier or spherical harmonics angular bases. The inverse methods are `coef_i()` and `ylm_i()`.

Mappings
Class Map_af



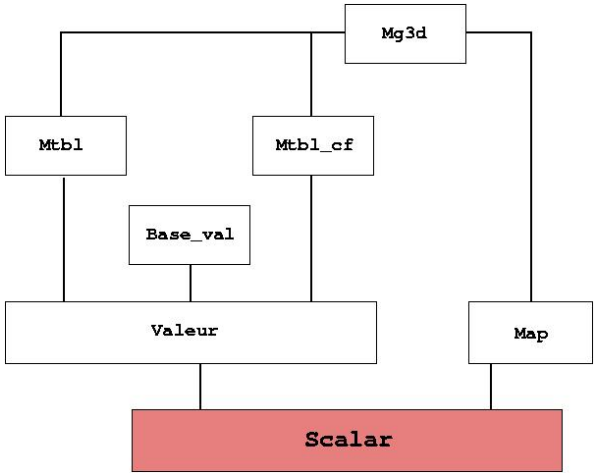


A mapping relates, in each domain, the numerical grid coordinates (ξ, θ', φ') to the physical ones (r, θ, φ).

The simplest class is `Map_af` for which the relation between ξ and r is linear (nucleus + shells) or inverse (CED).

To a mapping are attached coordinate fields `Coord` :
$r, \theta, \varphi, x, y, z, \cos\theta, \cdots$; vector orthogonal triads and flat metrics.

Scalar fields
Class Scalar



The class `Scalar` gathers a `Valeur` and a mapping, it represents a scalar field defined on the spectral grid, or a component of a vector/tensor.

A way to construct a `Scalar` is to

1. use the standard constructor, which needs a mapping ; the associated `Valeur` being then constructed in an undefined state (`ETATNONDEF` ;
2. assign it an expression using `Coords` : *e.g.* `x*y + exp(z)`.

Important methods of the class Scalar

Accessors and modifier of the Valeur

- `get_spectral_va()` readonly
- `set_spectral_va()` read/write ; it can be used to compute spectral coefficients, or to access directly to the coefficients (`Mtbl_cf`).

Spectral base manipulation

- `std_spectral_base()` sets the standard spectral base for a scalar field ;
- `std_spectral_base_odd()` sets the spectral base for the radial derivative of a scalar field ;
- `get_spectral_base()` returns the `Base_val` of the considered `Scalar` ;
- `set_spectral_base(Base_val)` sets a given `Base_val` as the spectral base.

Important methods of the class Scalar

Accessors and modifier of values in a given domain

- `domain(int)` reading;
- `set_domain(int)` modifying; it can be used to change the values in the physical space in one domain only.

Accessors and modifier of values of a grid point

- `val_grid_point(int, int, int, int)` readonly in the physical space;
- `set_grid_point(int, int, int, int)` read/write in the physical space, but should be used with caution, read carefully the documentation.

The dzpuis flag

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

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

For the inversion Laplace operator, since

$$\Delta_r = u^4 \Delta_u,$$

it is interesting to have the source multiplied by r^4 in the CED.
$\Rightarrow$use of an integer flag `dzpuis` for a scalar field f, which means that in the CED, one does not have f, but

$$r^{\text{dzpuis}} f$$

stored.
For instance, if f is constant equal to one in the CED, but with a `dzpuis` set to 4, it means that $\boxed{f = 1/r^4}$ in the CED.

Regular operators and finite part

An operator like $1/r^2$ is singular, in general, at the origin.
Nevertheless, when it appears within *e.g.* the Laplace operator

$$\Delta = \frac{\partial^2}{\partial r^2} + \frac{2}{r}\frac{\partial}{\partial r} + \frac{1}{r^2}\Delta_{\theta\varphi}$$

it should give regular results, when applied to a regular field.
$\Rightarrow$parity $+$ r^ℓ behavior near the origin ensure that everything is well behaved...in theory !
In practice, numerical errors can make things diverge if the division by r is performed in the physical space.
$\Rightarrow$these kind of operators are evaluated in the coefficient space, resulting in

$$\frac{1}{r} \leftrightarrow \frac{f(r) - f(0)}{r}.$$

Operator matrices

All radial operators can be seen, in a given domain, as a matrix multiplication on the vector of Chebyshev coefficients.
The class `Diff` and its derived classes can give directly this matrix :

- there is a different type for each operator
- for example, the second derivative is `Diff_dsdx2`
- standard constructors for all these classes need the number of coefficients and the type of spectral base :

```
Diff_dsdx2 op(17, R_CHEBP) ;
const Matrice mat_op = op.get_matrice() ;
```

Note that this gives the operator with respect to the ξ coordinate...

Vector fields

Lorene can handle a vector field $\boldsymbol{V}$ (class `Vector`) expressed in either of two types of components (*i.e.* using two *orthonormal* triads, of type `Base_vect`) :

- the spherical triad $(V_r, V_\theta, V_\varphi)$ `get_bvect_spher()`,
- the Cartesian triad (V_x, V_y, V_z) `get_bvect_cart()`.

Note that the choice of triad is independent from that of coordinates : one can use $V_y(r, \theta, \varphi)$.

- The Cartesian components of a regular vector field in spherical coordinates follow the same rules that a regular scalar field, except for symmetries ;
- The spherical components have more complicated rules since the spherical triad is singular (additional singularity).

$\Rightarrow$two ways of defining a regular vector field in spherical components :

- define it in Cartesian components and then rotate it (method `change_triad(Base_vect)`), or
- define it as a gradient of a regular scalar field.