The SageManifolds project

Differential geometry with a computer

Éric Gourgoulhon

Laboratoire Univers et Théories (LUTH)
CNRS / Observatoire de Paris / Université Paris Diderot
92190 Meudon, France

http://luth.obspm.fr/~luthier/gourgoulhon/

based on a collaboration with
Michał Bejger, Marco Mancini, Travis Scrimshaw

Institut de recherche mathématique de Rennes
Université Rennes I
13 May 2015

Outline

Outline

Introduction

- Computer algebra system (CAS) started to be developed in the 1960's; for instance `Macsyma` (to become `Maxima` in 1998) was initiated in 1968 at MIT

Introduction

- Computer algebra system (CAS) started to be developed in the 1960's; for instance `Macsyma` (to become `Maxima` in 1998) was initiated in 1968 at MIT
- In 1965, J.G. Fletcher develop the `GEOM` program, to compute the Riemann tensor of a given metric

Introduction

- Computer algebra system (CAS) started to be developed in the 1960's; for instance `Macsyma` (to become `Maxima` in 1998) was initiated in 1968 at MIT
- In 1965, J.G. Fletcher develop the `GEOM` program, to compute the Riemann tensor of a given metric
- In 1969, during his PhD under Pirani supervision at King's College, Ray d'Inverno wrote `ALAM (Atlas Lisp Algebraic Manipulator)` and used it to compute the Riemann tensor of Bondi metric. The original calculations took Bondi and his collaborators 6 months to go. The computation with ALAM took 4 minutes and yield to the discovery of 6 errors in the original paper [J.E.F. Skea, *Applications of SHEEP* (1994)]

Introduction

- Computer algebra system (CAS) started to be developed in the 1960's; for instance `Macsyma` (to become `Maxima` in 1998) was initiated in 1968 at MIT
- In 1965, J.G. Fletcher develop the `GEOM` program, to compute the Riemann tensor of a given metric
- In 1969, during his PhD under Pirani supervision at King's College, Ray d'Inverno wrote `ALAM (Atlas Lisp Algebraic Manipulator)` and used it to compute the Riemann tensor of Bondi metric. The original calculations took Bondi and his collaborators 6 months to go. The computation with ALAM took 4 minutes and yield to the discovery of 6 errors in the original paper [J.E.F. Skea, *Applications of SHEEP* (1994)]
- In the early 1970's, ALAM was rewritten in the LISP programming language, thereby becoming machine independent and renamed `LAM`

Introduction

- Computer algebra system (CAS) started to be developed in the 1960's; for instance `Macsyma` (to become `Maxima` in 1998) was initiated in 1968 at MIT
- In 1965, J.G. Fletcher develop the `GEOM` program, to compute the Riemann tensor of a given metric
- In 1969, during his PhD under Pirani supervision at King's College, Ray d'Inverno wrote `ALAM (Atlas Lisp Algebraic Manipulator)` and used it to compute the Riemann tensor of Bondi metric. The original calculations took Bondi and his collaborators 6 months to go. The computation with ALAM took 4 minutes and yield to the discovery of 6 errors in the original paper [J.E.F. Skea, *Applications of SHEEP* (1994)]
- In the early 1970's, ALAM was rewritten in the LISP programming language, thereby becoming machine independent and renamed `LAM`
- The descendant of LAM, called `SHEEP` (!), was initiated in 1977 by Inge Frick

Introduction

- Computer algebra system (CAS) started to be developed in the 1960's; for instance `Macsyma` (to become `Maxima` in 1998) was initiated in 1968 at MIT
- In 1965, J.G. Fletcher develop the `GEOM` program, to compute the Riemann tensor of a given metric
- In 1969, during his PhD under Pirani supervision at King's College, Ray d'Inverno wrote `ALAM (Atlas Lisp Algebraic Manipulator)` and used it to compute the Riemann tensor of Bondi metric. The original calculations took Bondi and his collaborators 6 months to go. The computation with ALAM took 4 minutes and yield to the discovery of 6 errors in the original paper [J.E.F. Skea, *Applications of SHEEP* (1994)]
- In the early 1970's, ALAM was rewritten in the LISP programming language, thereby becoming machine independent and renamed `LAM`
- The descendant of LAM, called `SHEEP` (!), was initiated in 1977 by Inge Frick
- Since then, many softwares for tensor calculus have been developed...

Software for differential geometry

Packages for general purpose computer algebra systems:

- xAct free package for Mathematica [J.-M. Martin-Garcia]
- Ricci free package for Mathematica [J. L. Lee]
- MathTensor package for Mathematica [S. M. Christensen & L. Parker]
- DifferentialGeometry included in Maple [I. M. Anderson & E. S. Cheb-Terrab]
- Atlas 2 for Maple and Mathematica
- ...

Standalone applications:

- SHEEP, Classi, STensor, based on Lisp, developed in 1970's and 1980's (free) [R. d'Inverno, I. Frick, J. Åman, J. Skea, et al.]
- Cadabra field theory (free) [K. Peeters]
- SnapPy topology and geometry of 3-manifolds, based on Python (free) [M. Culler, N. M. Dunfield & J. R. Weeks]
- ...

cf. the complete list at http://www.xact.es/links.html

Sage in a few words

- Sage (SageMath) is a **free open-source** mathematics software system
- it is based on the Python programming language
- it makes use of **many pre-existing open-sources packages**, among which
  - Maxima (symbolic calculations, since 1968!)
  - GAP (group theory)
  - PARI/GP (number theory)
  - Singular (polynomial computations)
  - matplotlib (high quality 2D figures)

  and provides a **uniform interface** to them
- William Stein (Univ. of Washington) created Sage in 2005; since then, ∼**100 developers** (mostly mathematicians) have joined the Sage team

The mission

Create a viable free open source alternative to Magma, Maple, Mathematica and Matlab.

Some advantages of Sage

Sage is free

Freedom means

1. everybody can use it, by downloading the software from http://sagemath.org
2. everybody can examine the source code and improve it

Sage is based on Python

- no need to learn any specific syntax to use it
- easy access for students
- Python is a very powerful *object oriented language*, with a neat syntax

Sage is developing and spreading fast

...sustained by an important community of developers

Object-oriented notation in Python

As an object-oriented language, Python (and hence Sage) makes use of the following **postfix notation** (same in C++, Java, etc.):

```
result = object.function(arguments)
```

In a procedural language, this would be written as

```
result = function(object,arguments)
```

Object-oriented notation in Python

As an object-oriented language, Python (and hence Sage) makes use of the following **postfix notation** (same in C++, Java, etc.):

```
result = object.function(arguments)
```

In a procedural language, this would be written as

```
result = function(object,arguments)
```

Examples

1. `riem = g.riemann()`
2. `lie_t_v = t.lie_der(v)`

NB: no argument in example 1

Sage approach to computer mathematics

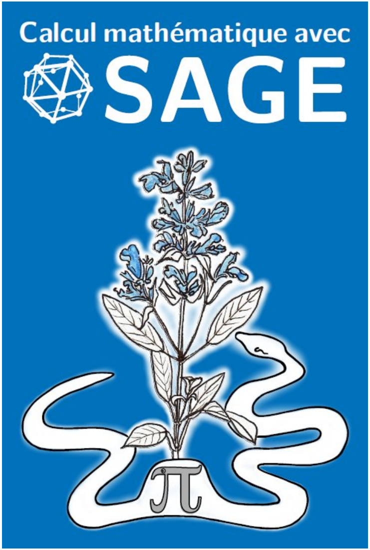
Sage relies on a `Parent` / `Element` scheme: each object x on which some calculus is performed has a "parent", which is another Sage object X representing the set to which x belongs.
The calculus rules on x are determined by the *algebraic structure* of X.
Conversion rules prior to an operation, e.g. $x + y$ with x and y having different parents, are defined at the level of the parents

Example

```
sage: x = 4 ; x.parent()
Integer Ring
sage: y = 4/3 ; y.parent()
Rational Field
sage: s = x + y ; s.parent()
Rational Field
sage: y.parent().has_coerce_map_from(x.parent())
True
```

This approach is similar to that of Magma and is different from that of Mathematica, in which everything is a tree of symbols

The Sage book



by A. Casamayou, N. Cohen, G. Connan, T. Dumont, L. Fousse, F. Maltey, M. Meulien, M. Mezzarobba, C. Pernet, N.M. Thiéry & P. Zimmermann (2013)

Released under *Creative Commons* license:

- freely downloadable from http://sagebook.gforge.inria.fr/
- printed copies can be ordered at moderate price (10 €)

Differential geometry in Sage

Sage is well developed in many domains of mathematics but not too much in the area of differential geometry:

Already in Sage

- differential forms on an open subset of Euclidean space *(with a fixed set of coordinates)* (J. Vankerschaver)
- parametrized 2-surfaces in 3-dim. Euclidean space (M. Malakhaltsev, J. Vankerschaver, V. Delecroix)
- hyperbolic geometry (models of $\mathbb{H}^2$ without explicitly specifying the metric) (G.Laun, V. Delecroix, M. Raum) (since Sage 6.6)

Outline

The SageManifolds project

http://sagemanifolds.obspm.fr/

Aim

Implement real smooth manifolds of arbitrary dimension in Sage and tensor calculus on them, in a coordinate/frame-independent manner

In particular:

- one should be able to introduce an arbitrary number of coordinate charts on a given manifold, with the relevant transition maps
- tensor fields must be manipulated as such and not through their components with respect to a specific (possibly coordinate) vector frame

The SageManifolds project

http://sagemanifolds.obspm.fr/

Aim

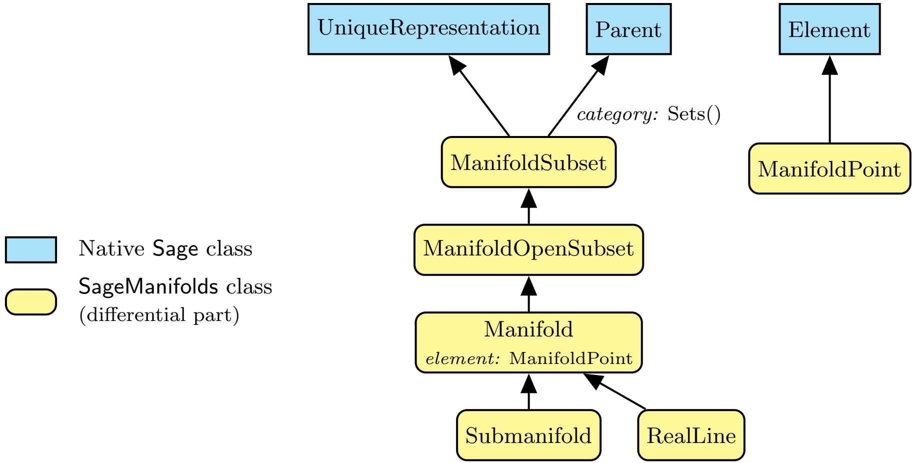
Implement real smooth manifolds of arbitrary dimension in Sage and tensor calculus on them, in a coordinate/frame-independent manner

In particular:

- one should be able to introduce an arbitrary number of coordinate charts on a given manifold, with the relevant transition maps
- tensor fields must be manipulated as such and not through their components with respect to a specific (possibly coordinate) vector frame

Concretely, the project amounts to creating new Python classes, such as `Manifold`, `Chart`, `TensorField` or `Metric`, within Sage's **Parent/Element framework**.

Implementating manifolds and their subsets



Implementing coordinate charts

Given a (topological) manifold M of dimension $n \geq 1$, a **coordinate chart** is a homeomorphism $\varphi : U \to V$, where U is an open subset of M and V is an open subset of $\mathbb{R}^n$.
Coordinate harts are implemented in SageManifolds via the class `Chart`, whose main data is U and a n-tuple of *Sage symbolic variables* `x`, `y`, ..., each of them representing a coordinate.

Implementing coordinate charts

Given a (topological) manifold M of dimension $n \geq 1$, a **coordinate chart** is a homeomorphism $\varphi : U \to V$, where U is an open subset of M and V is an open subset of $\mathbb{R}^n$.
Coordinate harts are implemented in SageManifolds via the class `Chart`, whose main data is U and a n-tuple of *Sage symbolic variables* `x`, `y`, ..., each of them representing a coordinate.

In general, more than one chart is required to cover the entire manifold:

Examples:

- at least 2 charts are necessary to cover the n-dimensional sphere $\mathbb{S}^n$ $(n \geq 1)$ and the torus $\mathbb{T}^2$
- at least 3 charts are necessary to cover the real projective plane $\mathbb{RP}^2$

Implementing coordinate charts

Given a (topological) manifold M of dimension $n \geq 1$, a **coordinate chart** is a homeomorphism $\varphi : U \to V$, where U is an open subset of M and V is an open subset of $\mathbb{R}^n$.
Coordinate harts are implemented in SageManifolds via the class `Chart`, whose main data is U and a n-tuple of *Sage symbolic variables* `x`, `y`, ..., each of them representing a coordinate.

In general, more than one chart is required to cover the entire manifold:

Examples:

- at least 2 charts are necessary to cover the n-dimensional sphere $\mathbb{S}^n$ $(n \geq 1)$ and the torus $\mathbb{T}^2$
- at least 3 charts are necessary to cover the real projective plane $\mathbb{RP}^2$

In SageManifolds, an arbitrary number of charts can be introduced

To fully specify the manifold, one shall also provide the *transition maps* on overlapping chart domains (SageManifolds class `CoordChange`)

Implementing scalar fields

A **scalar field** on manifold M is a smooth mapping

$$\begin{array}{llll} f: & U \subset M & \longrightarrow & \mathbb{R} \\ & p & \longmapsto & f(p) \end{array}$$

where U is an open subset of M.

Implementing scalar fields

A **scalar field** on manifold M is a smooth mapping

$$\begin{array}{rccl} f: & U \subset M & \longrightarrow & \mathbb{R} \\ & p & \longmapsto & f(p) \end{array}$$

where U is an open subset of M.

A scalar field maps *points*, not *coordinates*, to real numbers
$\Longrightarrow$ an object f in the `ScalarField` class has different **coordinate representations** in different charts defined on U.

Implementing scalar fields

A **scalar field** on manifold M is a smooth mapping

$$\begin{array}{rccl} f: & U \subset M & \longrightarrow & \mathbb{R} \\ & p & \longmapsto & f(p) \end{array}$$

where U is an open subset of M.

A scalar field maps *points*, not *coordinates*, to real numbers
$\Longrightarrow$ an object f in the `ScalarField` class has different **coordinate representations** in different charts defined on U.

The various coordinate representations F, $\hat{F}$, ... of f are stored as a *Python dictionary* whose keys are the charts C, $\hat{C}$, ...:

$$f.\mathtt{\_express} = \left\{ C : F,\ \hat{C} : \hat{F}, \ldots \right\}$$

with $f(\underbrace{p}_{\text{point}}) = F(\underbrace{x^1, \ldots, x^n}_{\text{coord. of } p \text{ in chart } C}) = \hat{F}(\underbrace{\hat{x}^1, \ldots, \hat{x}^n}_{\text{coord. of } p \text{ in chart } \hat{C}}) = \ldots$

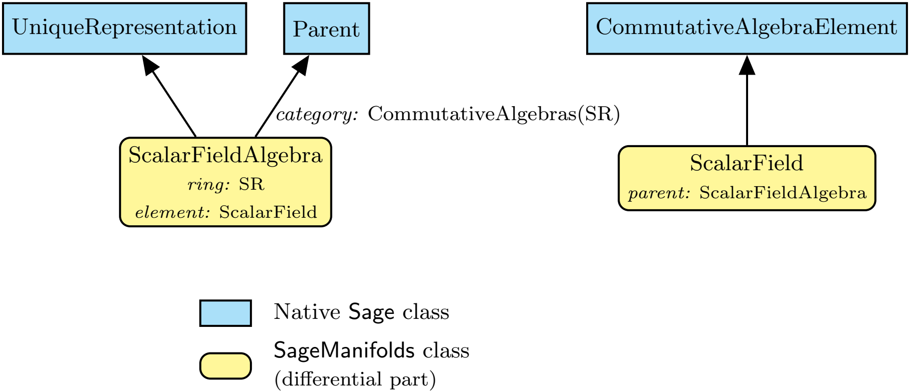
The scalar field algebra

Given an open subset $U \subset M$, the set $C^\infty(U)$ of scalar fields defined on U has naturally the structure of a **commutative algebra over** $\mathbb{R}$: it is clearly a vector space over $\mathbb{R}$ and it is endowed with a commutative ring structure by pointwise multiplication:

$$\forall f, g \in C^\infty(U), \quad \forall p \in U, \quad (f.g)(p) := f(p)g(p)$$

The algebra $C^\infty(U)$ is implemented in SageManifolds via the class `ScalarFieldAlgebra`.

Classes for scalar fields



Vector field modules

Given an open subset $U \subset M$, the set $\mathcal{X}(U)$ of smooth vector fields defined on U has naturally the structure of a **module over the scalar field algebra** $C^\infty(U)$.

$\mathcal{X}(U)$ is a free module $\iff$ U admits a global vector frame $(\boldsymbol{e}_a)_{1\leq a\leq n}$:

$$\forall \boldsymbol{v} \in \mathcal{X}(U), \quad \boldsymbol{v} = v^a \boldsymbol{e}_a, \quad \text{with } v^a \in C^\infty(U)$$

At any point $p \in U$, the above translates into an identity in the *tangent vector space* T_pM:

$$\boldsymbol{v}(p) = v^a(p)\, \boldsymbol{e}_a(p), \quad \text{with } v^a(p) \in \mathbb{R}$$

Example:

If U is the domain of a coordinate chart $(x^a)_{1\leq a\leq n}$, $\mathcal{X}(U)$ is a free module of rank n over $C^\infty(U)$, a basis of it being the coordinate frame $(\partial/\partial x^a)_{1\leq a\leq n}$.

Parallelizable manifolds

M is a **parallelizable manifold** $\iff$ M admits a global vector frame
$\iff$ $\mathcal{X}(M)$ is a free module
$\iff$ M's tangent bundle is trivial: $TM \simeq M \times \mathbb{R}^n$

Parallelizable manifolds

M is a **parallelizable manifold** $\iff$ M admits a global vector frame
$\iff$ $\mathcal{X}(M)$ is a free module
$\iff$ M's tangent bundle is trivial: $TM \simeq M \times \mathbb{R}^n$

Examples of parallelizable manifolds

- $\mathbb{R}^n$ (global coordinate charts $\Rightarrow$ global vector frames)
- the circle $\mathbb{S}^1$ (NB: no global coordinate chart)
- the torus $\mathbb{T}^2 = \mathbb{S}^1 \times \mathbb{S}^1$
- the 3-sphere $\mathbb{S}^3 \simeq \mathrm{SU}(2)$, as any Lie group
- the 7-sphere $\mathbb{S}^7$
- any orientable 3-manifold (Steenrod theorem)

Parallelizable manifolds

M is a **parallelizable manifold** $\iff$ M admits a global vector frame
$\iff$ $\mathcal{X}(M)$ is a free module
$\iff$ M's tangent bundle is trivial: $TM \simeq M \times \mathbb{R}^n$

Examples of parallelizable manifolds

- $\mathbb{R}^n$ (global coordinate charts $\Rightarrow$ global vector frames)
- the circle $\mathbb{S}^1$ (NB: no global coordinate chart)
- the torus $\mathbb{T}^2 = \mathbb{S}^1 \times \mathbb{S}^1$
- the 3-sphere $\mathbb{S}^3 \simeq \mathrm{SU}(2)$, as any Lie group
- the 7-sphere $\mathbb{S}^7$
- any orientable 3-manifold (Steenrod theorem)

Examples of non-parallelizable manifolds

- the sphere $\mathbb{S}^2$ (hairy ball theorem!) and any n-sphere $\mathbb{S}^n$ with $n \notin \{1, 3, 7\}$
- the real projective plane $\mathbb{RP}^2$

Implementing vector fields

Ultimately, in SageManifolds, vector fields are to be described by their components w.r.t. various vector frames.

If the manifold M is not parallelizable, we assume that it can be covered by a finite number N of parallelizable open subsets U_i $(1 \leq i \leq N)$ (OK for M compact). We then consider **restrictions** of vector fields to these domains:

Implementing vector fields

Ultimately, in SageManifolds, vector fields are to be described by their components w.r.t. various vector frames.

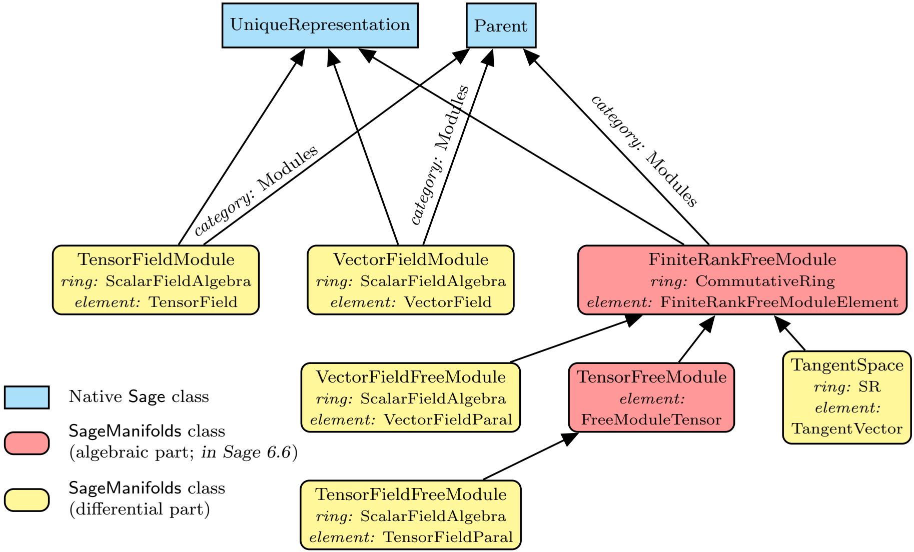
If the manifold M is not parallelizable, we assume that it can be covered by a finite number N of parallelizable open subsets U_i $(1 \leq i \leq N)$ (OK for M compact). We then consider **restrictions** of vector fields to these domains:

For each i, $\mathcal{X}(U_i)$ is a free module of rank $n = \dim M$ and is implemented in SageManifolds as an instance of `VectorFieldFreeModule`, which is a subclass of `FiniteRankFreeModule`.

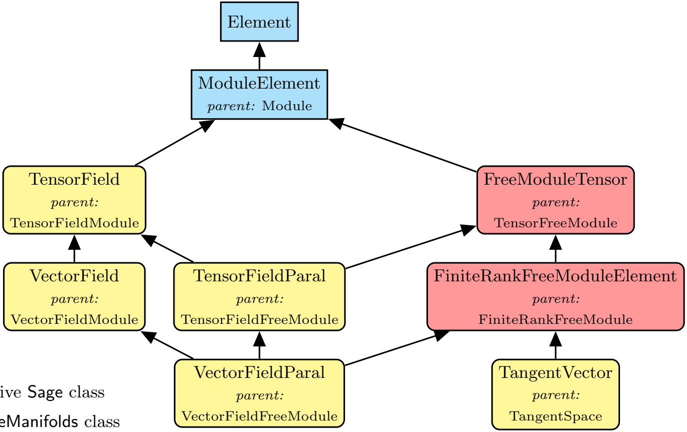
Each vector field $\boldsymbol{v} \in \mathcal{X}(U_i)$ has different set of components $(v^a)_{1 \leq a \leq n}$ in different vector frames $(\boldsymbol{e}_a)_{1 \leq a \leq n}$ introduced on U_i. They are stored as a *Python dictionary* whose keys are the vector frames:

$$\boldsymbol{v}.\texttt{\_components} = \{(\boldsymbol{e}) : (v^a),\ (\hat{\boldsymbol{e}}) : (\hat{v}^a), \ldots\}$$

Module classes in SageManifolds



Tensor field classes

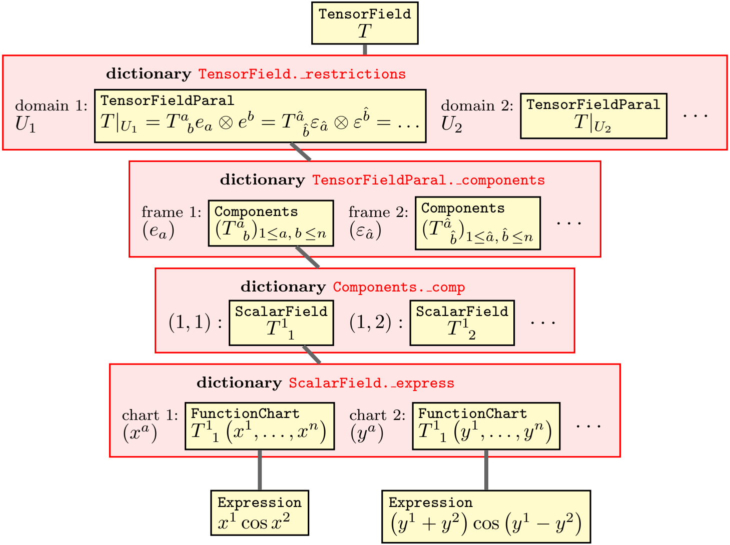


Native Sage class

SageManifolds class (algebraic part; *in Sage 6.6*)

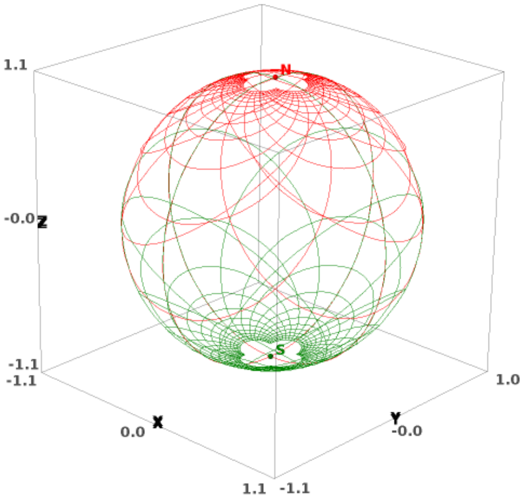
SageManifolds class (differential part)

Tensor field storage



Outline

The 2-sphere example



Function `Chart.plot()`

Stereographic coordinates on the 2-sphere

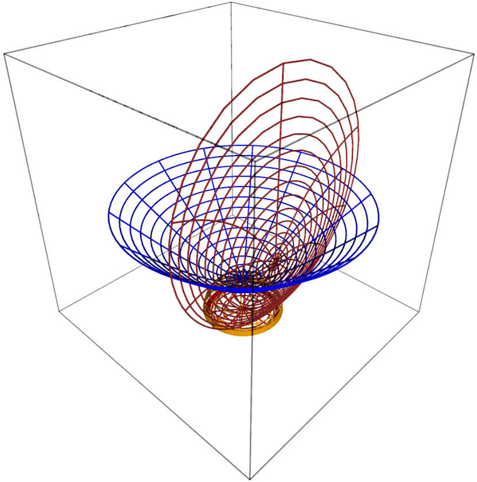
Two charts:

- X_1: $\mathbb{S}^2 \setminus \{N\} \to \mathbb{R}^2$
- X_2: $\mathbb{S}^2 \setminus \{S\} \to \mathbb{R}^2$

See the worksheet at

http://sagemanifolds.obspm.fr/examples/html/SM_sphere_S2.html

The hyperbolic plane example



Charts associated with various models of $\mathbb{H}^2$:

- hyperboloidal model (blue)
- Poincaré disk model (red)
- hemispherical model (orange)
- Poincaré half-plane model (brown)

$\Longrightarrow$ See the worksheet here

Outline

Conclusion and perspectives

- **SageManifolds** is a work in progress
  $\sim$ 47,000 lines of Python code up to now (including comments and doctests)
- A preliminary version (v0.7) is freely available (GPL) at
  `http://sagemanifolds.obspm.fr/`
  and the development version (to become v0.8 in a few days!) is available from the Git repository `https://github.com/sagemanifolds/sage`

Conclusion and perspectives

- **SageManifolds** is a work in progress
  $\sim$ 47,000 lines of Python code up to now (including comments and doctests)
- A preliminary version (v0.7) is freely available (GPL) at
  http://sagemanifolds.obspm.fr/
  and the development version (to become v0.8 in a few days!) is available from the Git repository https://github.com/sagemanifolds/sage

Example: installing SageManifolds 0.7 in a branch of a Sage 6.6 install

```
cd <your Sage root directory>
git remote add sm-github https://github.com/sagemanifolds/sage.git
git fetch -t sm-github sm-v0.7
git checkout -b sagemanifolds
git merge FETCH_HEAD
make
```

More details at http://sagemanifolds.obspm.fr/download.html

Current status

Already present (v0.7):

- maps between manifolds, pullback operator
- submanifolds, pushforward operator
- curves in manifolds
- standard tensor calculus (tensor product, contraction, symmetrization, etc.), even on non-parallelizable manifolds
- all monoterm tensor symmetries
- exterior calculus (wedge product, exterior derivative, Hodge duality)
- Lie derivatives of tensor fields
- affine connections, curvature, torsion
- pseudo-Riemannian metrics, Weyl tensor
- some plotting capabilities (charts, points, curves)

Current status

- *In the development version (v0.8 very soon):*
    - parallelization (on tensor components) of CPU demanding computations, via the Python library `multiprocessing`
    - graphical output for vector fields
    - textbook notations for partial derivatives of symbolic functions
    - nice outputs for tables of Christoffel symbols and tensor components
    - standard math operators ($\exp$, $\cos$, etc.) on scalar fields

Current status

- *In the development version (v0.8 very soon):*
  - parallelization (on tensor components) of CPU demanding computations, via the Python library `multiprocessing`
  - graphical output for vector fields
  - textbook notations for partial derivatives of symbolic functions
  - nice outputs for tables of Christoffel symbols and tensor components
  - standard math operators ($\exp$, $\cos$, etc.) on scalar fields
- *Not implemented yet (but should be soon):*
  - extrinsic geometry of pseudo-Riemannian submanifolds
  - computation of geodesics (numerical integration via Sage/GSL or `Gyoto`)
  - integrals on submanifolds

Current status

- *In the development version (v0.8 very soon):*
  - parallelization (on tensor components) of CPU demanding computations, via the Python library `multiprocessing`
  - graphical output for vector fields
  - textbook notations for partial derivatives of symbolic functions
  - nice outputs for tables of Christoffel symbols and tensor components
  - standard math operators ($\exp$, $\cos$, etc.) on scalar fields
- *Not implemented yet (but should be soon):*
  - extrinsic geometry of pseudo-Riemannian submanifolds
  - computation of geodesics (numerical integration via Sage/GSL or `Gyoto`)
  - integrals on submanifolds
- *Future prospects:*
  - add more graphical outputs
  - add more functionalities: symplectic forms, fibre bundles, spinors, variational calculus, etc.
  - connection with numerical relativity: using Sage to explore numerically-generated spacetimes

Integration into Sage

SageManifolds is aimed to be fully integrated into Sage

- The algebraic part (tensors on free modules of finite rank) has been submitted to Sage Trac as ticket #15916 and has got a positive review $\Longrightarrow$ integrated in Sage 6.6
- The differential part will be split in various tickets for submission to Sage Trac; meanwhile, one has to download it from `http://sagemanifolds.obspm.fr/`

Acknowledgements: the SageManifolds project has benefited from many discussions with Sage developers around the world, and especially in Paris area (V. Delecroix, M. Mezzarobba, T. Monteil, N. Thiéry)

Want to join the project or simply to stay tuned?

visit `http://sagemanifolds.obspm.fr/`
(download page, documentation, example worksheets, mailing list)