ACTAS DEL XVI CONGRESO
DR. ANTONIO A. R. MONTEIRO (2021)
2023, Páginas 67–116

GEOMETRÍA DE GRUPOS DE LIE USANDO SAGEMATH

SILVIO REGGIANI

RESUMEN. Introduciremos de manera amigable las nociones elementales de la geometría riemanniana (conexión de Levi-Civita, geodésicas, curvatura, isometrías, etc.). Para ejemplificar estos conceptos estudiaremos métricas invariantes a izquierda en grupos de Lie tridimensionales. Trataremos de mostrar con estos ejemplos cómo podemos ayudarnos de SageMath para resolver problemas de geometría riemanniana.

ÍNDICE

2020 *Mathematics Subject Classification.* Primary 53C05; Secondary 53C30.

1. Introducción

La idea de este curso es presentar, de manera informal, algunos de los conceptos básicos que se estudian en geometría riemanniana homogénea (esto incluye grupos y álgebras de Lie). A la par iremos viendo cómo se implementan estos conceptos en el software SageMath y algunas de sus funcionalidades.

En SageMath las variedades diferenciables están implementadas en el proyecto SageManifolds. Al momento de dictar el curso, la última versión disponible de SageMath era la versión 9.3. Sin embargo, al momento de terminar estas notas ya se cuenta con la versión 9.4. Estas notas están, dentro de lo posible, adaptadas a la última versión de SageMath. Las notebooks originales (así como algunas imágenes ilustrativas que no tienen la calidad suficiente como para ser incluidas en estas actas) se encuentran en el repositorio https://github.com/silvioreggiani/curso-monteiro.

[1]:
```
version()
```

[1]:
```
'SageMath version 9.4, Release Date: 2021-08-22'
```

Algunos enlaces relevantes para quienes quieran ejecutar las notebooks mencionadas más arriba son:

- https://doc.sagemath.org/html/en/installation/ para la instalación de SageMath y Jupyter.
- Alternativamente, las notebooks pueden ejecutarse en https://cocalc.com/ con una cuenta gratuita (la versión gratuita es un poco lenta, pero más que suficiente para este curso).
- Para quienes no estén familiarizados con SageMath, se recomienda hacer antes algún tutorial, por ejemplo https://doc.sagemath.org/html/en/tutorial/index.html
- Finalmente, https://sagemanifolds.obspm.fr/ es la página oficial del proyecto SageManifolds. Además de la documentación oficial, allí se pueden encontrar los cambios hechos en las últimas actualizaciones.

Con respecto a la parte matemática del curso, la exposición será autocontenida pero informal. Un curso de calculo en varias variables o geometría de curvas y superficies alcanzará como base. Para profundizar en los temas que cubriremos se recomienda estudiar los conceptos elementales de variedades diferenciables y riemannianas, esto puede encontrarse, por ejemplo, en los primeros capítulos del libro de do Carmo [dC92].

2. Variedades diferenciables

Decimos que un espacio topológico M es una *variedad diferenciable* de dimensión n si

- tenemos coordenadas locales $x_1, \dots, x_n$ en (un entorno abierto de) cada punto $p \in M$;
- el cambio de coordenadas $(x_1, \dots, x_n) \mapsto (y_1, \dots, y_n)$ es diferenciable (para nosotros diferenciablilidad significará de clase C^∞);
- la topología de M es *suficientemente buena* (espacio localmente euclídeo).

Para definir una variedad diferenciable en Sage tenemos el comando `Manifold()`. Por ejemplo, una variedad diferenciable M de dimensión 3 la indicamos como sigue.

```
[2]: M = Manifold(3, 'M'); M
```

```
[2]: 3-dimensional differentiable manifold M
```

Podemos pedirle a Sage que nos muestre los resultados usando LaTeX. Sin embargo, a veces el output mostrado de esta manera es demasiado largo para ser visualizado correctamente. Es por eso que para hacer la exposición más amena, alteremos entre el modo LaTeXy el modo código para mostrar los resultados.

```
[3]: %display latex
     M
```

[3]:

$$M$$

Investiguemos un poco qué es lo que acabamos de definir.

```
[4]: %display plain #mostramos los resultados en modo código
     M.parent()
```

```
[4]: <class 'sage.manifolds.differentiable.manifold.
       DifferentiableManifold_with_category'>
```

Lo anterior nos dice, rápidamente hablando, que `M` es un objeto de la clase de las variedades diferenciables. No nos preocuparemos mucho por saber qué es lo que significa un objeto y una clase. Solo diremos que esto viene heredado de Python, que es el software en el que SageMath está escrito. Otra manera, más matemática, de entender qué es `M` para Sage, es usando el método `.category()`.

```
[5]: %display latex #volvemos a mostrar el output con LaTeX
     M.category()
```

[5]:

$$\textbf{Smooth}_{\mathbf{R}}$$

Un mismo objeto puede estar en varias categorías.

```
[6]: %display plain
     M.categories()
```

```
[6]: [Category of smooth manifolds over Real Field with 53 bits of
       precision,
      Category of differentiable manifolds over Real Field with 53 bits
       of precision,
      Category of manifolds over Real Field with 53 bits of precision,
      Category of topological spaces,
      Category of sets,
      Category of sets with partial maps,
      Category of objects]
```

Observemos que todavía no tenemos definido ningún sistema de coordenadas en M, y por ende la lista

[7]:
```
%display latex
M.atlas()
```

[7]:

$$[\,]$$

es una lista vacía. Aquí, el método `.atlas()` busca los sistemas de coordenadas definidos por el usuario y no hay que confundirlo con el atlas maximal de una variedad diferenciable. Definamos una carta en M.

[8]:
```
x.<x1,x2,x3> = M.chart()
```

Con este comando hemos definido una carta global $x : M \to \mathbb{R}^3$ cuyas funciones coordenadas son x_1, x_2, x_3. (Una explicación un poco más técnica es que `.chart()` es un método de la clase `Manifold` que usamos para definir los sistemas de coordenadas.)

Ahora vemos que el atlas de M ya no está vacío.

[9]:
```
M.atlas()
```

[9]:

$$[(M, (x_1, x_2, x_3))]$$

Podemos definir un punto $p \in M$ especificando sus coordenadas en la carta que acabamos de definir y podemos mostrar p de diferentes maneras.

[10]:
```
p = M.point((1,-7,12), name='p')
print(p)
display(p)
display(p in M)
p.coord()
```

```
Point p on the 3-dimensional differentiable manifold M
```

$$p$$

True

[10]:

$$(1, -7, 12)$$

La variedad que acabamos de definir no es otra cosa que $\mathbb{R}^3$ ya que las coordenadas se mueven entre $-\infty$ y $+\infty$.

[11]:
```
x.coord_range()
```

[11]:

$$x_1 : (-\infty, +\infty); \quad x_2 : (-\infty, +\infty); \quad x_3 : (-\infty, +\infty)$$

En realidad, SageManifolds viene con un comando `EuclideanSpace()` que nos permite definir las variedades $\mathbb{R}^n$ de manera más sencilla.

[12]:
```
E3 = EuclideanSpace(3)
display(E3)
E3.atlas()
```

$$\mathbb{E}^3$$

[12]:

$$\left[\left(\mathbb{E}^3, (x,y,z)\right)\right]$$

Notar, sin embargo, que para Sage, `M` y `E3` son dos variedades distintas.

[13]:
```
M == E3
```

[13]:

False

De hecho, los espacios euclídeos vienen con estructuras adicionales, como por ejemplo la métrica riemanniana usual (inducida por el producto escalar). Observemos que el plano $\mathbb{E}^2$ viene también con coordenadas polares.

[14]:
```
E2 = EuclideanSpace(2)
E2.atlas()
```

[14]:

$$\left[\left(\mathbb{E}^2, (x,y)\right)\right]$$

Pero no vienen cargadas por defecto, tenemos que hacerlo manualmente.

[15]:
```
coord_cart = E2.cartesian_coordinates()
coord_polar = E2.polar_coordinates()
E2.atlas()
```

[15]:

$$\left[\left(\mathbb{E}^2, (x,y)\right), \left(\mathbb{E}^2, (r,\phi)\right)\right]$$

Naturalmente, $(r,\phi) \in \mathbb{R}^+ \times [0, 2\pi)$.

[16]:
```
coord_polar.coord_range()
```

[16]:

$$r:\ (0,+\infty);\quad \phi:\ [0,2\,\pi]\,\text{(periodic)}$$

Definamos un punto genérico en el plano y veamos el cambio de coordenadas.

[17]:
```
p = E2.point(coord_cart[:])
display(p.coord())
p.coord(coord_polar)
```

$$(x,y)$$

[17]:

$$\left(\sqrt{x^2+y^2}, \arctan(y,x)\right)$$

Observar que el método `.point()` usa la carta por defecto, que en nuestro caso son las coordenadas cartesianas.

[18]:
```
E2.default_chart()
```

[18]:

$$\left(\mathbb{E}^2, (x,y)\right)$$

Pero también podríamos definir un punto genérico en coordenadas polares. Como ya vimos antes el método `.coord()` también usa la carta por defecto, pero podemos pedirle que nos muestre las coordenadas con respecto a cualquier carta.

[19]:
```
q = E2.point(coord_polar[:],chart=coord_polar)
display(q.coord())
q.coord(coord_polar)
```

$$(r\cos(\phi), r\sin(\phi))$$

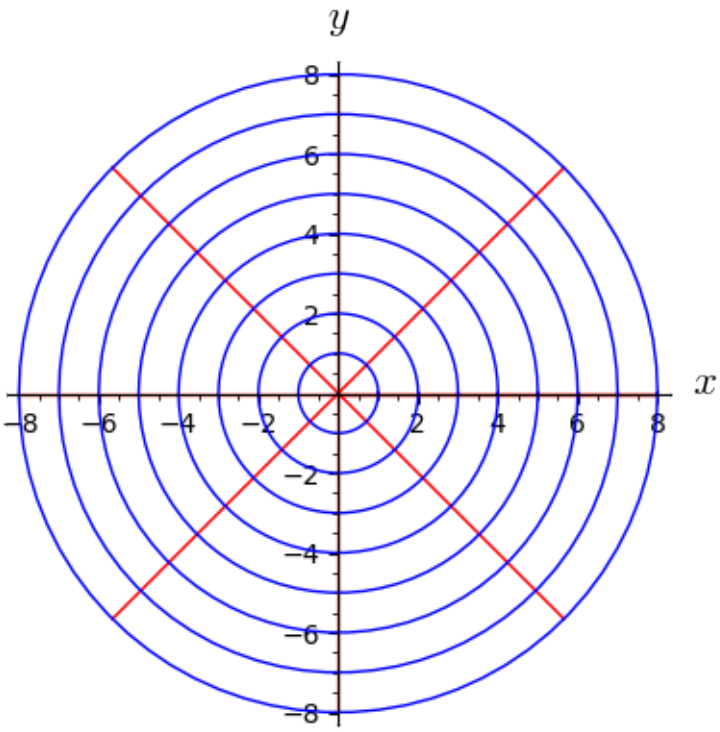
[19]:

$$(r, \phi)$$

Es posible graficar como se ve una carta coordenada con respecto a la otra.

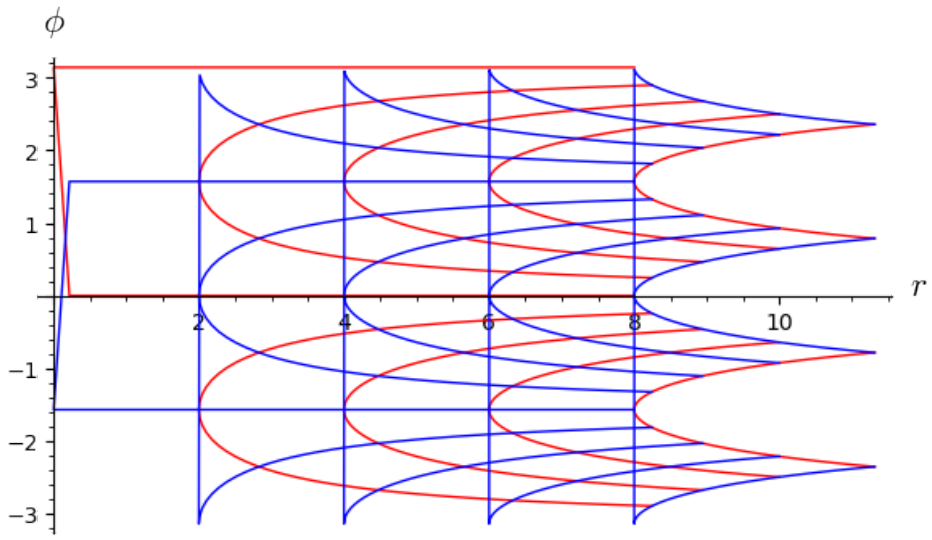
[20]:
```
coord_polar.plot(coord_cart,color={coord_polar[0]:'blue',
                                   coord_polar[1]:'red'}
                 )
```

[20]:



[21]:
```
coord_cart.plot(coord_polar,color={coord_cart[0]:'blue',
                                   coord_cart[1]:'red'}
                )
```

[21]:



Es interesante saber que el método `chart1.plot(chart2)` funciona en variedades de cualquier dimensión (obviamente, eligiendo secciones de dimensión 2 o 3 para mostrar los gráficos).

3. Funciones diferenciables

Si N y M son dos variedades diferenciables, una función $f : N \to M$ se dice *diferenciable* si lo es cuando la escribimos en coordenadas locales. Es decir, si $x_1, \ldots, x_n$ son coordenadas locales alrededor de $p \in N$ e $y_1, \ldots, y_m$ son coordenadas locales alrededor de $f(p) \in M$, podemos pensar localmente a f como una función $\mathbb{R}^n \to \mathbb{R}^m$ (y esta función es la que debe ser diferenciable).

Definamos la función $d : \mathbb{R}^2 \to \mathbb{R}$ que calcula el cuadrado de la distancia al origen. De paso, mostramos una forma distinta de definir la variedad diferenciable $\mathbb{R}$.

[22]:
```
R.<t> = manifolds.RealLine(); print(R)
```

```
Real number line ℝ
```

En la versión original de las notebooks usábamos directamente el comando `RealLine()`. En la nueva versión esto ya es obsoleto por lo que ahora debemos definir la recta usando `manifolds.RealLine()`.

[23]:
```
E2.<x,y> = EuclideanSpace(2); E2
```

[23]:

$$\mathbb{E}^2$$

Hemos vuelto a definir el plano, esta vez indicando el nombre de las coordenadas, lo cual nos resulta más cómodo para definir la función d. Para definir una función diferenciable de `M` en `N` podemos usar `M.diff_map(N, ...)`.

[24]:
```
d = E2.diff_map(R, x^2+y^2)
d.display()
```

[24]:

$$\begin{array}{ccc} \mathbb{E}^2 & \longrightarrow & \mathbf{R} \\ (x,y) & \longmapsto & t = x^2 + y^2 \end{array}$$

Luego d es un elemento de lo que usualmente se denota por $C^\infty(\mathbb{R}^2, \mathbb{R})$. En el lenguaje de categorías SageMath presenta esto de la siguiente forma:

[25]:
```
display(d.parent())
print(d.parent()) #verbose
```

$$\mathrm{Hom}\left(\mathbb{E}^2, \mathbf{R}\right)$$

```
Set of Morphisms from Euclidean plane E^2 to Real number line R in
Category of smooth connected manifolds over Real Field with 53 bits
of precision
```

Las funciones se pueden evaluar en puntos como lo haríamos normalmente.

[26]:
```
p = E2.point((-1,2))
d(p).coord()
```

[26]:

(5)

Los ejemplos hasta ahora son bastante simples porque las variedades que definimos son también sencillas. En general, para cubrir todos los puntos de una variedad diferenciable M hace falta más de una carta. En estos casos, para definir una función diferenciable f usando `diff_map()`, tendremos que pasar como argumentos las expresiones en coordenadas de f con respecto a un subconjunto de cartas que cubra toda la variedad M. En breve veremos ejemplos de esto también.

4. Curvas y vectores tangentes

Una *curva* en M es una función diferenciable

$$c : I \to M$$

en donde I es un intervalo de $\mathbb{R}$. La *velocidad* de la curva $c(t)$, con respecto a una carta $x_1, \ldots, x_n$ se define como

$$c'(t) \mapsto (c_1'(t), \ldots, c_n'(t))$$

en donde $c_1(t), \ldots, c_n(t)$ son las coordenadas de $c(t)$ en la carta antes mencionada.

Es posible definir la velocidad de una curva independientemente del sistema de coordenadas, considerando la clase de equivalencia de estos vectores en $\mathbb{R}^n$ si se puede pasar de uno a otro vía (la matriz jacobiana de) un cambio de coordenadas. Alternativamente, esto se puede hacer considerando a los vectores tangentes como derivaciones (aplicaciones lineales que cumplen la regla de Leibniz) del álgebra de funciones diferenciables de M.

A modo de ejemplo una espiral de Fermat en el plano. Recordemos que en coordenadas polares una espiral de Fermat, que empieza en el origen, tiene la forma

$$r = \pm a\sqrt{\phi}, \qquad \phi \geq 0$$

[27]:
```
R.<t> = manifolds.RealLine()
E2 = EuclideanSpace(2)
```

```
coord_cart.<x,y> = E2.cartesian_coordinates()
coord_polar.<r,ph> = E2.polar_coordinates()
```

Algunas aclaraciones sobre lo que sigue. En lugar de definir la curva usando el método `R.diff_map(E2, ...)` es preferible hacerlo con el método `E2.curve(...)`. Igualmente definimos la recta `R.<t> = manifolds.RealLine()` para que Sage cree la variable simbólica `t`. Otra forma de hacer esto sería con `t = var('t')`. Ahora sí podemos definir nuestra curva.

[28]:
```
c = E2.curve((t,t^2), (t,0,3),chart=coord_polar)
print(c)
c.display()
```

```
Curve in the Euclidean plane E^2
```

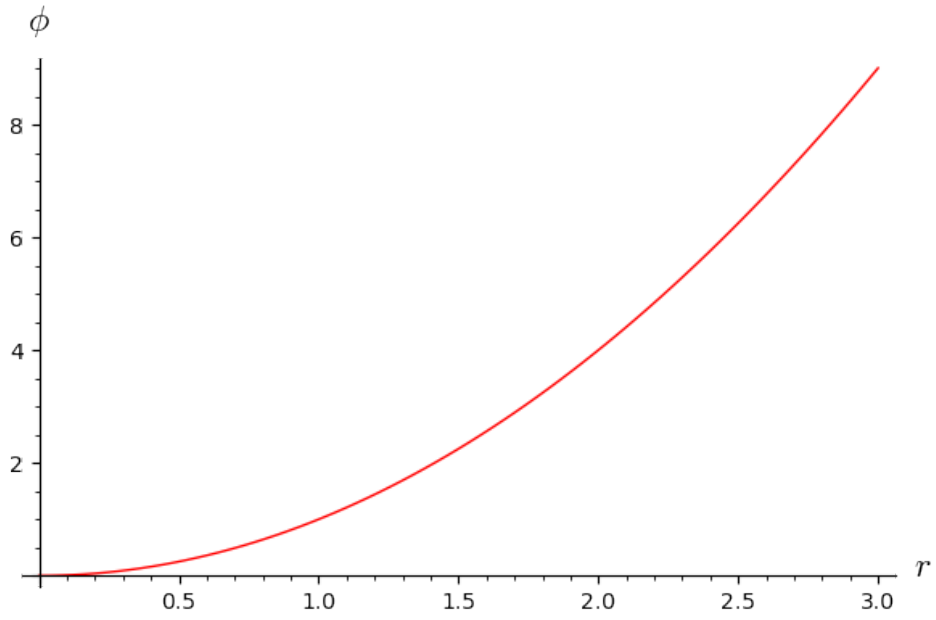
[28]:

$$\begin{array}{lcl} (0,3) & \longrightarrow & \mathbb{E}^2 \\ t & \longmapsto & (x,y) = \left(t\cos\left(t^2\right), t\sin\left(t^2\right)\right) \\ t & \longmapsto & (r,\phi) = \left(t, t^2\right) \end{array}$$

Observemos que SageMath nos muestra la expresión de $c(t)$ en los dos sistemas de coordenadas disponibles. Grafiquemos la espiral de Fermat, primero en coordenadas polares:

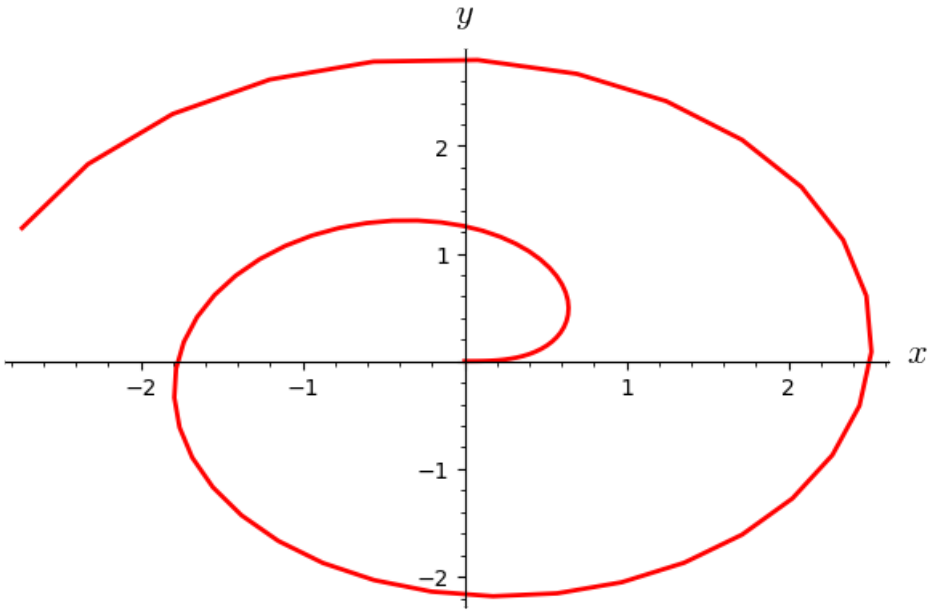
[29]:
```
c.plot(coord_polar)
```

[29]:



Y luego en coordenadas cartesianas:

[30]:
```
c1 = E2.curve(c.coord_expr(coord_cart), (t,0,3))
g1 = c1.plot(thickness=2)
show(g1)
```

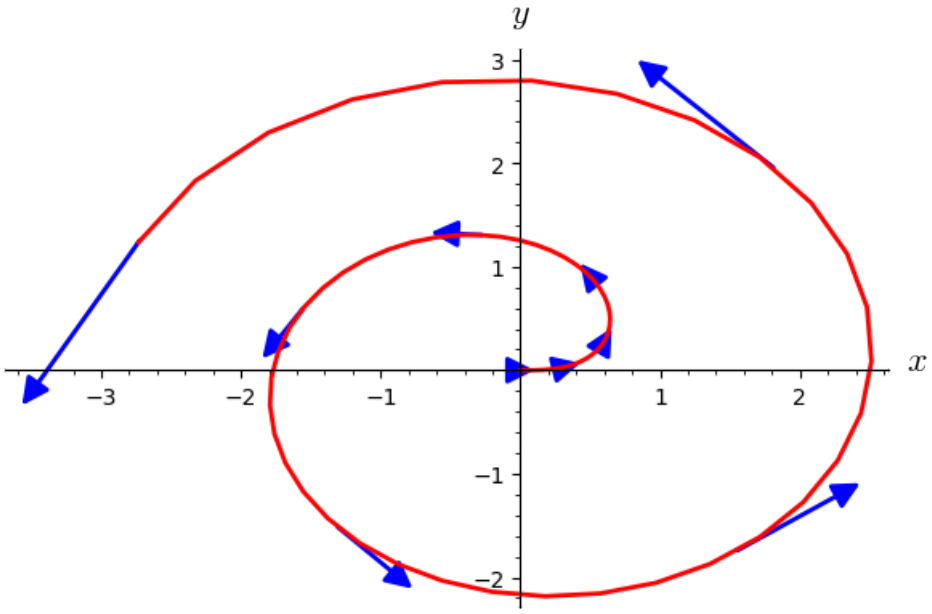


Finalmente definamos y grafiquemos la velocidad de esta curva.

[31]:
```
v = c1.tangent_vector_field()
print(v)
display(v.display())
g2 = v.plot(chart=coord_cart, number_values=10,scale=0.1)
show(g1+g2)
```

```
Vector field along the Real interval (0, 3) with values on the
  Euclidean plane E^2
```

$$\left(-2t^2\sin\left(t^2\right)+\cos\left(t^2\right)\right)e_x+\left(2t^2\cos\left(t^2\right)+\sin\left(t^2\right)\right)e_y$$



5. ESPACIO TANGENTE

Si $p \in M$, el *espacio tangente* a M en p se define por

$$T_pM = \{v = c'(0) : c \text{ curva en } M \text{ tal que } c(0) = p\}.$$

Dado $v \in T_pM$, podemos calcular la *derivada direccional* de una función $f : M \to \mathbb{R}$ en la dirección de v por:

$$v(f) = \frac{d}{dt}\Big|_0 f(c(t))$$

en donde $c(t)$ es una curva en M tal que $c(0) = p$ y $c'(0) = v$. Esta definición no depende de la curva elegida (siempre que tenga velocidad v). Más aún, las derivadas direccionales son independientes de las coordenadas locales. Esto último es muy importante, porque nos dice que el espacio vectorial T_pM existe en cada punto y se puede "construir" independientemente de las coordenadas locales.

Veamos cómo se trabaja con los espacios tangentes en Sage. Primero definimos un punto en una variedad de dimensión 4.

[32]:
```
M = Manifold(4, 'M')
M_chart.<x,y,z,w> = M.chart()
p = M.point((1,0,-5,1), name='p')
```

Y usamos el método `.tangent_space()` para acceder a T_pM. Observemos que debemos pasar como argumento el punto base.

[33]:
```
TpM = M.tangent_space(p)
print(TpM)
TpM
```

```
Tangent space at Point p on the 4-dimensional differentiable
  manifold M
```

[33]:

$$T_pM$$

Como T_pM es un espacio vectorial, está en muchas categorías:

[34]:
```
%display plain
TpM.categories()
```

[34]:
```
[Category of finite dimensional vector spaces over Symbolic Ring,
 Category of finite dimensional modules over Symbolic Ring,
 Category of vector spaces over Symbolic Ring,
 Category of modules over Symbolic Ring,
 Category of bimodules over Symbolic Ring on the left and Symbolic
   Ring on the right,
 Category of right modules over Symbolic Ring,
 Category of left modules over Symbolic Ring,
 Category of commutative additive groups,
 Category of additive groups,
 Category of additive inverse additive unital additive magmas,
 Category of commutative additive monoids,
 Category of additive monoids,
 Category of additive unital additive magmas,
 Category of commutative additive semigroups,
 Category of additive commutative additive magmas,
 Category of additive semigroups,
 Category of additive magmas,
 Category of sets,
 Category of sets with partial maps,
 Category of objects]
```

Hay varios métodos interesantes para un espacio vectorial en SageMath. Por ejemplo,

[35]:
```
%display latex
display(TpM.dim())
display(TpM.default_basis())
vp = TpM.an_element(); vp.display()
```

4

$$\left(\frac{\partial}{\partial x}, \frac{\partial}{\partial y}, \frac{\partial}{\partial z}, \frac{\partial}{\partial w}\right)$$

[35]:

$$\frac{\partial}{\partial x} + 2\frac{\partial}{\partial y} + 3\frac{\partial}{\partial z} + 4\frac{\partial}{\partial w}$$

En la notación clásica el vector `vp` debería escribirse como

$$\left.\frac{\partial}{\partial x}\right|_p + 2\left.\frac{\partial}{\partial y}\right|_p + 3\left.\frac{\partial}{\partial z}\right|_p + 4\left.\frac{\partial}{\partial w}\right|_p.$$

Aquí esto no es necesario, ya que al ser `vp` un elemento de `TpM`, SageMath puede inferir que `p` es el punto base. Solamente deberemos tener un poco de cuidado de no confundirnos. Definamos otro vector tangente

[36]:
```
wp = TpM._element_constructor_([5,-2,1,0])
wp.display()
```

[36]:

$$5\frac{\partial}{\partial x} - 2\frac{\partial}{\partial y} + \frac{\partial}{\partial z}$$

y veamos que efectivamente los vectores tangentes se pueden sumar.

[37]:
```
display(vp + wp)
(vp + wp).display()
```

Tangent vector at Point p on the 4-dimensional differentiable manifold M

[37]:

$$6\frac{\partial}{\partial x} + 4\frac{\partial}{\partial z} + 4\frac{\partial}{\partial w}$$

Ahora veamos cómo calcular las derivadas direccionales. Técnicamente hablando, en Sage podemos derivar, a lo largo de un vector tangente, *campos escalares* definidos sobre M, en lugar de funciones $M \to \mathbb{R}$. Desde el punto de vista matemático, estos dos conceptos coinciden, pero para Sage `M.diff_map(R,...)` y `M.scalar_field()` son cosas distintas.

[38]:
```
R.<t> = manifolds.RealLine()
f = M.scalar_field(4*x*y + z^2 - cos(w*y))
f.display()
```

[38]:

$$\begin{array}{lcl} M & \longrightarrow & \mathbb{R} \\ (x,y,z,w) & \longmapsto & 4xy + z^2 - \cos(wy) \end{array}$$

Para calcular la derivada direccional simplemente aplicamos el vector v al campo escalar f.

[39]:
```
vp(f)
```

[39]:

$$-22$$

Ahora veamos que $v(f)$ coincide con $\frac{d}{dt}\big|_0 f(c(t))$ para una curva con $c(0) = p$ y $c'(0) = v$. Primero recordemos quiénes eran p y v.

[40]:
```
display(p.coord())
vp.display()
```

$$(1, 0, -5, 1)$$

[40]:

$$\frac{\partial}{\partial x}+2\frac{\partial}{\partial y}+3\frac{\partial}{\partial z}+4\frac{\partial}{\partial w}$$

Luego, necesitamos una curva que en el origen pase por $(1,0,-5,1)$ con velocidad $(1,2,3,4)$.

[41]:
```
c = M.curve((1 + t, 2*t , -5 + 3*t, 1 + 4*t), t)
print(c)
c.display()
```

```
Curve in the 4-dimensional differentiable manifold M
```

[41]:

$$\begin{array}{rcl} \mathbf{R} & \longrightarrow & M \\ t & \longmapsto & (x,y,z,w) = (t+1, 2t, 3t-5, 4t+1) \end{array}$$

Efectivamente

[42]:
```
c(0) == p
```

[42]:

True

Observemos que para calcular $c'(0)$ tenemos que evaluar el campo de velocidades en el punto `R(0)` (no lo podemos evaluar directamente en 0).

[43]:
```
vc = c.tangent_vector_field()
vc.at(R(0)) == vp
```

[43]:

True

Finalmente, hacemos

[44]:
```
%display plain
f(c(t))
```

[44]:
```
2*cos(t)^2*sin(4*t^2)^2 + 4*cos(4*t^2)*cos(t)*sin(4*t^2)*sin(t)
 + 2*cos(4*t^2)^2*sin(t)^2 + 17*t^2 - 22*t + 24
```

[45]:
```
f(c(t)).diff(t) #derivada con respecto a t
```

[45]:
```
32*t*cos(4*t^2)*cos(t)^2*sin(4*t^2) + 32*t*cos(4*t^2)^2*cos(t)*sin(t)
 - 32*t*cos(t)*sin(4*t^2)^2*sin(t)
 - 32*t*cos(4*t^2)*sin(4*t^2)*sin(t)^2
 + 4*cos(4*t^2)*cos(t)^2*sin(4*t^2) + 4*cos(4*t^2)^2*cos(t)*sin(t)
 - 4*cos(t)*sin(4*t^2)^2*sin(t) - 4*cos(4*t^2)*sin(4*t^2)*sin(t)^2
 + 34*t - 22
```

[46]:
```
f(c(t)).diff(t).subs(t=0) #evaluada en t = 0
```

[46]:
```
-22
```

6. Diferencial de una función

Al igual que en análisis en varias variables, la *diferencial* de una función diferenciable $f : M \to N$ en el punto $p \in M$ es la transformación lineal

$$\mathrm{d}f|_p : T_pM \to T_{f(p)}N$$

que *mejor aproxima* a f. Podemos definir la diferencial usando curvas: si $v = c'(0) \in T_pM$, entonces

$$\mathrm{d}f|_p(v) = \frac{d}{dt}\bigg|_0 f(c(t))$$

También podemos trabajar en coordenadas: si $x_1, \ldots, x_m$ son coordenadas locales en $p \in M$; $y_1, \ldots, y_m$ son coordenadas locales en $f(p) \in N$ y $f_1, \ldots, f_n$ es la expresión de f con respecto a este par de cartas, entonces la matriz de $\mathrm{d}f|_p$ con respecto a las bases $\frac{\partial}{\partial x_i}\big|_p$ y $\frac{\partial}{\partial y_j}\big|_{f(p)}$ es la matriz jacobiana

$$\left(\frac{\partial f_i}{\partial x_j}\bigg|_p\right)_{i,j}$$

Sage nos permite calcular la diferencial de una función en un punto dado usando el método `f.differential(p)`. A modo de ejemplo calculemos la diferencial del cambio de coordenadas polares a cartesianas en el plano.

[47]:
```
%display latex
R2_polar = Manifold(2, r'\mathbb R^2_\text{polar}')
polar_range = r'r:(0,+oo) theta:[0,2*pi]:periodic'
R2_polar_chart.<r,theta> = R2_polar.chart(polar_range)
R2_comun = Manifold(2, r'\mathbb R^2')
R2_comun_chart.<x,y> = R2_comun.chart()
f = R2_polar.diff_map(R2_comun,
                      (r*cos(theta), r*sin(theta)),
                      name='f'
                     )
f.display()
```

[47]:

$$\begin{array}{rccc} f: & \mathbb{R}^2_{\text{polar}} & \longrightarrow & \mathbb{R}^2 \\ & (r,\theta) & \longmapsto & (x,y) = (r\cos(\theta), r\sin(\theta)) \end{array}$$

Primero lo hacemos en un punto específico.

[48]:
```
p = R2_polar.point((1,pi/4),name='p')
fp = f(p)
p.coord(), fp.coord()
```

[48]:

$$\left(\left(1, \frac{1}{4}\pi\right), \left(\frac{1}{2}\sqrt{2}, \frac{1}{2}\sqrt{2}\right)\right)$$

Ahora calculamos la diferencial

[49]:
```
dfp = f.differential(p)
display(dfp)
print(dfp)
```

```
dfp.parent()
dfp.matrix()
```

$$\mathrm{d}f_p$$

```
Generic morphism:
  From: Tangent space at Point p on the 2-dimensional differentiable
    manifold \mathbb R^2_\text{polar}
  To:   Tangent space at Point f(p) on the 2-dimensional
    differentiable manifold \mathbb R^2
```

[49]:

$$\begin{pmatrix} \frac{1}{2}\sqrt{2} & -\frac{1}{2}\sqrt{2} \\ \frac{1}{2}\sqrt{2} & \frac{1}{2}\sqrt{2} \end{pmatrix}$$

Hagamos lo mismo pero en un punto genérico.

[50]:
```
r0,theta0 = var("r0 theta0")
assume(r0>0)
p = R2_polar.point((r0,theta0),name='p')
print(p)
display(p.coord())
dfp = f.differential(p)
dfp.matrix()
```

```
Point p on the 2-dimensional differentiable manifold
  \mathbb R^2_\text{polar}
```

$$(r_0, \theta_0)$$

[50]:

$$\begin{pmatrix} \cos(\theta_0) & -r_0\sin(\theta_0) \\ \sin(\theta_0) & r_0\cos(\theta_0) \end{pmatrix}$$

7. Campos vectoriales

Un *campo vectorial* X en M asigna a cada $p \in M$ un vector tangente $X_p \in T_pM$ de manera "suave". Si $x_1, \ldots, x_n$ son coordenadas locales tenemos, asociados n campos coordenados (locales)

$$\frac{\partial}{\partial x_1}, \frac{\partial}{\partial x_2}, \ldots, \frac{\partial}{\partial x_n}$$

los cuales se corresponden con los campos constantes $e_1, e_2, \ldots, e_n$ en $\mathbb{R}^n$. Los campos coordenados son linealmente independientes en cada punto y por ende generan cada espacio tangente en su dominio de definición: si $v \in T_pM$,

$$v = \sum_{i=1}^{n} v_i \frac{\partial}{\partial x_i}\bigg|_p$$

Si $f : M \to \mathbb{R}$,

$$v(f) = \sum_{i=1}^{n} v_i \frac{\partial f}{\partial x_i}\bigg|_p, \qquad \text{en donde } \frac{\partial f}{\partial x_i} = \frac{d}{dt} f(c_i(t)).$$

Aquí $c_i(t)$ es la curva cuyas coordenadas son $(x_1(p),\ldots,x_i(p)+t,\ldots,x_n(p))$. De esto último se desprende que $v_i = v(x_i)$.

Nuevamente hacemos un ejemplo en dimensión 4.

[51]:
```
M = Manifold(4, 'M')
M_chart.<x,y,z,w> = M.chart()
R.<t> = manifolds.RealLine()
f = M.scalar_field(4*x*y + z^2 - cos(w*y))
```

Para acceder a los campos coordenados de una carta local tenemos el método `.frame()`.

[52]:
```
M_chart.frame()
```

[52]:

$$\left(M, \left(\frac{\partial}{\partial x}, \frac{\partial}{\partial y}, \frac{\partial}{\partial z}, \frac{\partial}{\partial w}\right)\right)$$

Llamemos e_0, e_1, e_2, e_3 a estos campos coordenados y usemos esto para definir un nuevo campo X.

[53]:
```
e0,e1,e2,e3 = M_chart.frame()
X = x*e0 - e1 + z*w*e3; X
```

[53]:

Vector field on the 4-dimensional differentiable manifold M

Notemos que `X` aún no tiene un nombre en LaTeX, pero esto lo podemos resolver rápidamente con `.set_name()`.

[54]:
```
X.set_name('X')
display(X)
X.display()
```

$$X$$

[54]:

$$X = x\frac{\partial}{\partial x} - \frac{\partial}{\partial y} + wz\frac{\partial}{\partial w}$$

Ahora calculemos la derivada direccional $X(f)$.

[55]:
```
display(f)
display(f.display())
display(X(f))
X(f).disp()
```

Scalar field on the 4-dimensional differentiable manifold M

$$\begin{array}{lcl} M & \longrightarrow & \mathbb{R} \\ (x,y,z,w) & \longmapsto & 4xy + z^2 - \cos(wy) \end{array}$$

Scalar field on the 4-dimensional differentiable manifold M

[55]:

$$\begin{array}{lcl} M & \longrightarrow & \mathbb{R} \\ (x,y,z,w) & \longmapsto & wyz\sin(wy) + 4xy - w\sin(wy) - 4x \end{array}$$

También podemos usar el método `X(f).expr()` para acceder a la expresión simbólica de $X(f)$, por si necesitamos usarla en otro lado.

[56]:
```
display(X(f).expr())
print(X(f).expr())
```

$$wyz\sin(wy)+4xy-w\sin(wy)-4x$$

```
w*y*z*sin(w*y) + 4*x*y - w*sin(w*y) - 4*x
```

Lo siguiente también puede ser útil si en algún momento necesitamos reciclar alguna expresión en LaTeX.

[57]:
```
print(latex(X(f).expr()))
```

```
w y z \sin(w y) + 4 \, x y - w \sin(w y) - 4 \, x
```

8. Flujo de campos vectoriales

Todo campo vectorial $X \in \mathfrak{X}(M)$ se mueve a lo largo de unas curvas llamadas *curvas integrales* o *líneas de flujo* cuya velocidad es exactamente el valor del campo en el punto dado. Más precisamente, una *curva integral* de X por $p \in M$ es una curva $\gamma_p : I \subset \mathbb{R} \to M$ tal que $\gamma_p(0) = p$ y $\gamma_p'(t) = X_{\gamma_p(t)}$ para todo $t \in I$. Si pensamos ahora que t está fijo y p es quien varía obtenemos una familia de difeomorfismos locales

$$\varphi_t : U \subset M \to M, \qquad \varphi_t(p) = \gamma_p(t)$$

El conjunto $\{\varphi_t\}$ forma un grupo (local) llamado el *flujo* de X ya que

$$\varphi_0 = \mathrm{id}_M \qquad \varphi_t \circ \varphi_s = \varphi_{t+s} \qquad (\varphi_t)^{-1} = \varphi_{-t}$$

El flujo local de un campo X puede pensarse como un grupo de simetrías de X, pues

$$(\varphi_t)_*(X) = X$$

para todo $t \in I$.

9. Corchete de campos vectoriales y álgebras de Lie

El conjunto $\mathfrak{X}(M)$ de todos los campos vectoriales en M forma un un $C^\infty(M)$-módulo. En efecto, ya que los campos vectoriales se pueden sumar y multiplicar por funciones diferenciables. El corchete de Lie en $\mathfrak{X}(M)$ es una de las formas que tenemos para derivar un campo en la dirección de otro. Si $X = \sum a_i \frac{\partial}{\partial x_i}$, $Y = \sum b_j \frac{\partial}{\partial x_j}$, definimos el *corchete de Lie* entre X e Y como

$$[X,Y] = XY - YX = \sum_i \sum_j \left(a_j \frac{\partial b_i}{\partial x_j} - b_j \frac{\partial a_i}{\partial x_j} \right) \frac{\partial}{\partial x_i}$$

Notemos que la definición de $[X,Y]$ es independiente del sistema de coordenadas, pues se tiene que aplicado una función $f \in C^\infty(M)$, vale

$$[X,Y](f) = X(Y(f)) - Y(X(f)).$$

El espacio vectorial (sobre $\mathbb{R}$) $\mathfrak{X}(M)$ junto con el corchete de Lie forma un *álgebra de Lie* real (de dimensión infinita), ya que se cumplen

- $[\,,\,]$ es $\mathbb{R}$-bilineal.
- Antisimetría: $[X,Y] = -[Y,X]$.
- Identidad de Jacobi: $[X,[Y,Z]] + [Y,[Z,X]] + [Z,[X,Y]] = 0$.

Además se satisface la regla de Leibniz: si $f \in C^\infty(M)$,

$$[X, fY] = X(f)Y + f[X,Y]$$

El siguiente resultado nos dice que el corchete de Lie se puede usar para detectar cuando dos campos son campos coordenadas (con respecto a alguna carta coordenada).

Teorema. *Dos campos X e Y conmutan (i.e. $[X,Y]=0$) si y sólo si sus flujos conmutan $\varphi_t^X \circ \varphi_s^Y = \varphi_s^Y \circ \varphi_t^X$.*

Corolario (Teorema de Clairaut)**.** *Los campos coordenados conmutan.*

Demostración. En realidad esto es inmediato de la definición del corchete, pero hagámoslo usando flujos a modo de ejemplo. Como este es un teorema local, es suficiente probarlo para $M = \mathbb{R}^n$ y los campos coordenados $\frac{\partial}{\partial x_i} = e_i$. Aquí podemos calcular explícitamente $\varphi_t^i(p) = te_i + p$ y por ende $\varphi_t^i(\varphi_s^j(p)) = te_i + se_j + p = \varphi_s^j(\varphi_t^i(p))$. □

Verifiquemos con Sage que los campos coordenados conmutan.

[58]:
```
M = Manifold(3, 'M')
M_chart.<x,y,z> = M.chart()
M_frame = M_chart.frame()
M_frame
{X.bracket(Y).is_zero() for X in M_frame for Y in M_frame}
```

[58]:
```
{True}
```

En general dos campos cualesquiera no son conmutativos.

[59]:
```
X, Y, Z = M_frame
X0, X1, X2 = (x^2 + y^2 + z^2+1)*X, (y^2 + sin(z))*Y, exp(-x*2)*Z
X0.set_name('X_0'), X1.set_name('X_1'), X2.set_name('X_2')
display(X0.bracket(X1).display())
display(X0.bracket(X2).display())
display(X1.bracket(X2).display())
```

$$[X_0, X_1] = \left(-2y^3 - 2y\sin(z)\right)\frac{\partial}{\partial x}$$

$$[X_0, X_2] = -2ze^{(-2x)}\frac{\partial}{\partial x} - 2\left(x^2+y^2+z^2+1\right)e^{(-2x)}\frac{\partial}{\partial z}$$

$$[X_1, X_2] = -\cos(z)\,e^{(-2x)}\frac{\partial}{\partial y}$$

Los campos X_0, X_1, X_2 son linealmente independientes en cada punto y por lo tanto forman una base de cada espacio tangente. Esto se conoce como un *frame*. Sage nos permite definir frames que no necesariamente estén inducidos por un sistema de coordenadas. Esto será muy importante cuando trabajemos con grupos de Lie.

[60]:
```
M_other_frame = M.vector_frame('X', [X0, X1, X2])
M.frames()
```

[60]:

$$\left[\left(M,\left(\frac{\partial}{\partial x},\frac{\partial}{\partial y},\frac{\partial}{\partial z}\right)\right),(M,(X_0,X_1,X_2))\right]$$

Sin embargo, solo tenemos definido un sistema de coordenadas

[61]:
```
M.atlas()
```

[61]:

$$[(M,(x,y,z))]$$

Sage chequea que el frame que queremos definir sea linealmente independiente en cada punto (aunque no es bueno confiarse demasiado).

[62]:
```
M.vector_frame('X', [X0, X1, X0+X1])
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
/tmp/ipykernel_25603/2563714674.py in <module>
----> 1 M.vector_frame('X', [X0, X1, X0+X1])

/usr/lib/python3.9/site-packages/sage/manifolds/differentiable/
  manifold.py in vector_frame(self, *args, **kwargs)
   3152                             "input matrix must be nonsingular"]
   3153             if linked:
-> 3154         raise ValueError("the provided vector fields are not
   3155                                       "linearly independent")
   3156         # Adding the newly generated changes of frame to the

ValueError: the provided vector fields are not linearly independent
```

10. Grupos de Lie

Un *grupo de Lie* G es a la vez un grupo y una variedad diferenciable tal que las operaciones

$$\begin{aligned} m : G\times G &\to G & \qquad i : G &\to G \\ (x,y) &\mapsto xy & x &\mapsto x^{-1} \end{aligned}$$

son diferenciables. Algunos ejemplos importantes son:

- El grupo lineal general $GL_n(\mathbb{R})$ de matrices invertibles $n\times n$. Notar que $GL_n(\mathbb{R}) = \det^{-1}(\mathbb{R}-\{0\})$ es un grupo de Lie, ya que es un abierto en $M_{n\times n}(\mathbb{R})\simeq\mathbb{R}^{n^2}$. Muchos (casi todos) ejemplos de grupos de Lie se pueden presentar como subgrupos del grupo lineal general.
- El grupo ortogonal $O_n = \{A\in GL_n(\mathbb{R}) : AA^T = I_n\}$ el cual es un subgrupo compacto de $GL_n(\mathbb{R})$.
- El grupo lineal especial $SL_n(\mathbb{R}) = \{A\in GL_n(\mathbb{R}) : \det A = 1\}$.

10.1. Ejemplo: $SL_2(\mathbb{R})$. El grupo lineal especial $SL_2(\mathbb{R})$ es muy importante en muchas áreas de la matemática. Si quisiéramos probar que $SL_2(\mathbb{R})$ es una variedad diferenciable con *lápiz y papel* usaríamos el *teorema de la función implícita* para variedades (que también nos da información sobre su topología). Para definir en Sage las coordenadas de una matriz $A \in SL_2(\mathbb{R})$ lo haremos *a mano* escribiendo una de las entradas de A como función de las otras tres. En primer lugar, escribimos $A = \begin{pmatrix} a & b \\ c & d \end{pmatrix}$. Como $ad - bc = 1$, tenemos que $a \neq 0$ implica $d = \frac{1+bc}{a}$ y $b \neq 0$ implica $c = \frac{ad-1}{b}$. Además $SL_2(\mathbb{R}) = U \cup V$ con $U \simeq \{(a,b,c) \in \mathbb{R}^3 : a \neq 0\}$ y $V \simeq \{(a,b,d) \in \mathbb{R}^3 : b \neq 0\}$. Los símbolos $\simeq$ los hemos usado para indicar difeomorfismo (una función diferenciable biyectiva con inversa diferenciable).

Veamos nuestro primer ejemplo de una variedad diferenciable que no podemos cubrir con un solo sistema de coordenadas. En primer lugar, debemos definir los abiertos U y V usando el método `.open_subset()` y luego indicamos que con estos dos abiertos ya tenemos todos los puntos de $SL_2(\mathbb{R})$ usando `.declare_union()`.

[63]:
```
SL2 = Manifold(3, name=r"SL_2(\mathbb R)")
U = SL2.open_subset("U")
V = SL2.open_subset("V")
SL2.declare_union(U, V)
display(U.union(V))
display(SL2.open_cover_family())
```

$$SL_2(\mathbb{R})$$

$$\{\{SL_2(\mathbb{R})\},\{U,V\}\}$$

Ahora definimos las coordenadas en U y V como hicimos más arriba.

[64]:
```
rest_a = lambda a00,a01,a10: a00!=0
rest_b = lambda b00,b01,b11: b01!=0
a.<a00,a01,a10> = U.chart(coord_restrictions=rest_a)
b.<b00,b01,b11> = V.chart(coord_restrictions=rest_b)
```

Vale la pena aclarar que el parámetro `coord_restrictions=` es usado precisamente para indicar las coordenadas que queremos que sean no nulas en U y en V. En versiones anteriores de SageMath esto se hacía con el método `.add_restrictions()` que ahora es obsoleto. Ahora tenemos un atlas con dos sistemas de coordenadas:

[65]:
```
SL2.atlas()
```

[65]:

$$[(U,(a_{00},a_{01},a_{10})),(V,(b_{00},b_{01},b_{11}))]$$

Pero Sage aún piensa que U y V son abiertos disjuntos. Para resolver esto tenemos que definir el cambio de coordenadas. Si en la matriz $A = \begin{pmatrix} a & b \\ c & d \end{pmatrix}$ tenemos que $a \neq 0$ y $b \neq 0$ tendremos que

$$\begin{aligned} a_{00}(A) &= a, & a_{01}(A) &= b, & a_{10}(A) &= c, \\ b_{00}(A) &= a, & b_{01}(A) &= b, & b_{11}(A) &= d. \end{aligned}$$

Luego

$$b_{00} = a_{00}, \qquad b_{01} = a_{01}, \qquad b_{11} = \frac{1 + a_{01}a_{10}}{a_{00}}.$$

Para explicitar cómo son las coordenadas de `b` como función de las coordenadas de `a` usamos `a.transition_map(b, ...)`

[66]:
```
a_to_b = a.transition_map(b,
                          (a00, a01, (1+a01*a10)/a00),
                          intersection_name='W',
                          restrictions1= [a00!=0,a01!=0],
                          restrictions2 = [b00!=0,b01!=0]
                          )
a_to_b.display()
```

[66]:

$$\begin{cases} b_{00} &= a_{00} \\ b_{01} &= a_{01} \\ b_{11} &= \frac{a_{01}a_{10}+1}{a_{00}} \end{cases}$$

Para el otro cambio de coordenadas podemos usar el método `inverse()`.

[67]:
```
b_to_a = a_to_b.inverse()
b_to_a.display()
```

[67]:

$$\begin{cases} a_{00} &= b_{00} \\ a_{01} &= b_{01} \\ a_{10} &= \frac{b_{00}b_{11}-1}{b_{01}} \end{cases}$$

Ahora aparecen nuevos sistemas de coordenadas en nuestro atlas, porque en la intersección de los dominios de dos cartas coordenadas, tenemos definidas dos tipos de coordenadas.

[68]:
```
SL2.atlas()
```

[68]:

$$[(U,(a_{00},a_{01},a_{10})),(V,(b_{00},b_{01},b_{11})),(W,(a_{00},a_{01},a_{10})),(W,(b_{00},b_{01},b_{11}))]$$

Veamos un ejemplo de un elemento de `SL2` que no está en dicha intersección.

[69]:
```
A = SL2.point((0,16,-7),chart=b)
display(A in U)
display(A in V)
```

False

True

Podemos armar un pequeño programa que nos dé la representación matricial de los elementos de `SL2`.

[70]:
```
def SL2_to_mat(M):
    if M in U:
```

```
        M00,M01,M10 = M.coord(chart=a)
        M11 = (1+M01*M10)/M00
    else:
        M00,M01,M11 = M.coord(chart=b)
        M10 = (M00*M11-1)/M01
    return matrix(2,2,[M00,M01,M10,M11])
```

[71]:
```
A_mat = SL2_to_mat(A); A_mat
```

[71]:

$$\begin{pmatrix} 0 & 16 \\ -\frac{1}{16} & -7 \end{pmatrix}$$

Con la representación matricial podemos usar todos los métodos que Sage tiene definidos para matrices. Por ejemplo:

[72]:
```
display(A_mat.parent())
display(A_mat.det())
display(A_mat.characteristic_polynomial())
A_mat.inverse()
```

$$\mathrm{Mat}_{2\times 2}(\mathbf{Q})$$

$$1$$

$$x^2+7x+1$$

[72]:

$$\begin{pmatrix} -7 & -16 \\ \frac{1}{16} & 0 \end{pmatrix}$$

También podemos escribir un programa reciba matrices de determinante 1 y devuelva puntos de SL2.

[73]:
```
def mat_to_SL2(A):
    """
    Atención: este programa no chequea si A.det(1) == 1,
    tampoco chequea si los coeficientes de A son reales.
    """
    if A[0,0] != 0:
        return SL2.point((A[0,0],A[0,1],A[1,0]),chart=a)
    else:
        return SL2.point((A[0,0],A[0,1],A[1,1]),chart=b)
```

[74]:
```
A_mat = matrix(2,2,[0,1,-1,2]); A_mat
```

[74]:

$$\begin{pmatrix} 0 & 1 \\ -1 & 2 \end{pmatrix}$$

[75]:
```
A = mat_to_SL2(A_mat)
display(A in U)
display(A in V)
A.coord(b)
```

False

True

[75]:

$$(0,1,2)$$

Ejercicio. Pensar qué pasaría si hacemos `A.coord()`.

11. Campos invariantes a izquierda

Si bien, en general, los grupos de Lie no se pueden cubrir con una sola carta, siempre dispondremos de un frame global de campos *invariantes a izquierda*. Si G es un grupo de Lie, $e \in G$ es elemento neutro y $v \in T_eG$. El campo invariante a izquierda X^v inducido por v se define como

$$X^v|_g = \mathrm{d}L_g|_e v = \left.\frac{d}{dt}\right|_0 g \cdot c(t), \qquad g \in G,$$

en donde $c(0) = e$ y $c'(0) = v$. El corchete de Lie de dos campos invariantes a izquierda es nuevamente invariante a izquierda. Más aún,

$$\mathfrak{g} = \{X \in \mathfrak{X}(G) : X \text{ es invariante a izquierda}\}$$

es una subálgebra de Lie de dimensión finita $\dim \mathfrak{g} = \dim G$ de $\mathfrak{X}(G)$, llamada el *álgebra de Lie de* G. El álgebra de Lie es la versión infinitesimal de un grupo de Lie y todo grupo de Lie queda determinado localmente por su álgebra de Lie.

Tratemos de definir un campo invariante a izquierda en `SL2` (esta es una de esas cosas que es más fácil de hacer en papel que en Sage). En primer lugar, notemos que el álgebra de Lie $\mathfrak{sl}_2(\mathbb{R})$ de $SL_2(\mathbb{R})$ queda determinada por $T_I SL_2(\mathbb{R})$.

Ejercicio. La derivada en $t = 0$ de una curva de matrices de determinante 1 que en 0 pasa por I tiene traza 0. Es decir,

$$T_I SL_n(\mathbb{R}) = \{X \in M_{n\times n}(\mathbb{R}) : \mathrm{tr}\, A = 0\}.$$

Tomemos, por ejemplo,

$$v = c'(0) = \begin{pmatrix} 0 & 1 \\ 0 & 0 \end{pmatrix}, \qquad \text{donde } c(t) = \begin{pmatrix} 1 & t \\ 0 & 1 \end{pmatrix}.$$

Sea X el campo invariante a izquierda tal que $X_I = v$. Calculemos X_A con $A = \begin{pmatrix} a & b \\ c & d \end{pmatrix}$:

$$\left.\frac{d}{dt}\right|_0 A \cdot c(t) = \left.\frac{d}{dt}\right|_0 \begin{pmatrix} a & at+b \\ c & ct+d \end{pmatrix}$$

Para ver cuáles son las componentes de X tenemos que trabajar en coordenadas. Si $A \in U$,

$$\left.\frac{d}{dt}\right|_0 (a, at+b, c) = (0, a, 0).$$

Por otro lado, si $A \in V$,

$$\frac{d}{dt}\Big|_0(a, at+b, ct+d) = (0, a, c) = (0, a, (ad-1)/b).$$

Para hacer todo esto en Sage, primero inicializamos un campo `X`

[76]:
```
X = SL2.vector_field(name='X')
X.parent()
```

[76]:

$$\mathfrak{X}(SL_2(\mathbb{R}))$$

y luego indicamos cuáles son sus coeficientes según los campos coordenados:

[77]:
```
X[a.frame(),:] = [0,a00,0]
X[b.frame(),:] = [0,b00,(b00*b11-1)/b01]
display(X.display(a))
display(X.display(b))
```

$$X = a_{00}\frac{\partial}{\partial a_{01}}$$

$$X = b_{00}\frac{\partial}{\partial b_{01}} + \left(\frac{b_{00}b_{11}-1}{b_{01}}\right)\frac{\partial}{\partial b_{11}}$$

Por la forma en que lo definimos, ya sabemos que X es invariante a izquierda. A modo de ejemplo, verifiquemos con alguna matriz A que

$$X_A = \mathrm{d}L_A|_I(X_I)$$

[78]:
```
A = SL2.point((1,0,2),name='A')
display(SL2_to_mat(A))
display(A in U)
display(A in V)
```

$$\begin{pmatrix} 1 & 0 \\ 2 & 1 \end{pmatrix}$$

True

False

Para calcular $L_A : SL_2(\mathbb{R}) \to SL_2(\mathbb{R})$ necesitamos saber cuáles son las coordenadas de AB para cualquier $B \in SL_2(\mathbb{R})$. En este caso tenemos una pequeña ventaja, ya que $L_A(U) \subset U$ y $L_A(V) \subset V$.

Un punto genérico en U se puede obtener como sigue.

[79]:
```
assume(a00!=0,b01!=0) #para que las coordenadas sean válidas
SL2.point(a[:])
SL2_to_mat(A), SL2_to_mat(SL2.point(a[:]))
```

[79]:

$$\left(\begin{pmatrix} 1 & 0 \\ 2 & 1 \end{pmatrix}, \begin{pmatrix} a_{00} & a_{01} \\ a_{10} & \frac{a_{01}a_{10}+1}{a_{00}} \end{pmatrix}\right)$$

Ahora podemos multiplicar normalmente las matrices

[80]:
```
SL2_to_mat(A) * SL2_to_mat(SL2.point(a[:]))
```

[80]:

$$\begin{pmatrix} a_{00} & a_{01} \\ 2a_{00}+a_{10} & 2a_{01}+\frac{a_{01}a_{10}+1}{a_{00}} \end{pmatrix}$$

y luego volver a SL2 con mat_to_SL2:

[81]:
```
mat_to_SL2(SL2_to_mat(A) * SL2_to_mat(SL2.point(a[:]))).coord()
```

[81]:

$$(a_{00}, a_{01}, 2a_{00}+a_{10})$$

Una forma fácil de obtener las coordenadas en el formato apropiado es usar print() y luego copiarlas.

[82]:
```
print(_)
```

```
(a00, a01, 2*a00 + a10)
```

Hacemos lo mismo con los puntos de V (teniendo cuidado de usar las coordenadas de la carta b).

[83]:
```
print(mat_to_SL2(SL2_to_mat(A) * \
                 SL2_to_mat(SL2.point(b[:],b))).coord(b))
```

```
(b00, b01, 2*b01 + b11)
```

Ya estamos en condiciones de definir el difeomorfismo L_A ya que conocemos la expresión en todos los sistemas de coordenadas.

[84]:
```
LA = SL2.diff_map(SL2,{(a,a):(a00, a01, 2*a00 + a10),
             (b,b):(b00, b01, 2*b01 + b11)},name="L_A")
LA.display()
```

[84]:

$$\begin{array}{lccc} L_A: & SL_2(\mathbb{R}) & \longrightarrow & SL_2(\mathbb{R}) \\ \text{on } U: & (a_{00},a_{01},a_{10}) & \longmapsto & (a_{00},a_{01},2a_{00}+a_{10}) \\ \text{on } V: & (b_{00},b_{01},b_{11}) & \longmapsto & (b_{00},b_{01},2b_{01}+b_{11}) \end{array}$$

Definimos la matriz identidad.

[85]:
```
I2 = SL2.point((1,0,0),name='I_2')
display(SL2_to_mat(I2))
LA(I2)== A #chequeamos A * I2 = A
```

$$\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$$

[85]:

True

Calculamos la diferencial $\mathrm{d}L_A|_{I_2} : T_{I_2}SL_2(\mathbb{R}) \to T_ASL_2(\mathbb{R})$:

[86]:
```
dLA = LA.differential(I2)
display(dLA.parent())
dLA.matrix()
```

$$\mathrm{Hom}\left(T_{I_2} SL_2(\mathbb{R}), T_{L_A(I_2)} SL_2(\mathbb{R})\right)$$

[86]:

$$\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 2 & 0 & 1 \end{pmatrix}$$

Y finalmente chequeamos que $\mathrm{d}L_A|_{I_2} X_{I_2} = X_A$.

[87]:
```
dLA(X.at(I2)) == X.at(A)
```

[87]:

True

Ejercicio. 1) Escribir una función `left_translation(A)` que reciba un elemento $A \in SL_2(\mathbb{R})$ y devuelva la traslación a izquierda $L_A : SL_2(\mathbb{R}) \to SL_2(\mathbb{R})$.
2) Definir un frame invariante a izquierda en `SL2`.

12. Teoría de Lie en 2'

12.1. Teorema de Ado. Toda álgebra de Lie (real) de dimensión finita es isomorfa a una subálgebra de Lie de $\mathfrak{gl}_n(\mathbb{R})$ para algún n. Es decir, toda álgebra de Lie puede pensarse como una subálgebra de matrices con respecto al corchete

$$[A, B] = AB - BA$$

12.2. Álgebras de Lie semisimples y solubles. Hay dos clases distinguidas de álgebras de Lie de dimensión finita: las álgebras de Lie *(semi)simples* y las álgebras de Lie *solubles*. Las álgebras de Lie semisimples son las "más sencillas", están clasificadas y de esta clasificación (e.g. [Hel78]) sabemos que hay una cantidad finita en cada dimensión. Por otro lado las álgebras de Lie solubles pueden presentarse como subálgebras de matrices triangulares, aunque son bastante más complicadas de describir. No existe una clasificación completa y se sabe que hay familias infinitas de álgebras solubles no isomorfas en una misma dimensión (a partir de dimensión 3).

El llamado Teorema de descomposición de Levi nos dice que toda álgebra de Lie $\mathfrak{g}$ tiene una parte soluble y una parte semisimple que determinan completamente su estructura

$$\mathfrak{g} \simeq \mathfrak{g}_{\mathrm{sol}} \rtimes \mathfrak{g}_{\mathrm{ss}}$$

12.3. La aplicación exponencial. Si $G \subset GL_n(\mathbb{R})$ es un grupo de Lie y $\mathfrak{g} \subset \mathfrak{gl}_n(\mathbb{R})$ es su álgebra de Lie, la función exponencial nos da un difeomorfismo local entre un entorno del origen en $\mathfrak{g}$ y un entorno de la identidad en G.

$$\exp : \mathfrak{g} \to G$$
$$A \mapsto e^A = I_n + A + \frac{1}{2!}A^2 + \frac{1}{3!}A^3 + \cdots$$

Una familia muy importante de grupos/álgebras de Lie solubles son los grupos/álgebras de Lie *nilpotentes*. Si pensamos matricialmente un álgebra de Lie nilpotente, sus elementos se pueden representar con matrices triangulares estrictas. Una ventaja de los grupos de Lie solubles y, sobre todo, los nilpotentes (simplemente conexos) es que son mucho más fáciles de implementar en Sage, ya se pueden cubrir con una sola carta.

Teorema. 1. *Si G es soluble simplemente conexo, entonces G es difeomorfo a $\mathbb{R}^n$ con $n = \dim G$.*

2. *Si G es nilpotente simplemente conexo, entonces* $\exp : \mathfrak{g} \to G$ *es un difeomorfismo. Más aún, la serie de* e^A *es una suma finita para toda* $A \in \mathfrak{g}$.

Sage también viene con álgebras de Lie

[88]:
```
sl2 = lie_algebras.sl(QQ,2, representation='matrix')
print(sl2)
display(sl2.gens())
display(sl2.is_abelian())
display(sl2.is_semisimple())
display(sl2.is_solvable())
```

```
Special linear Lie algebra of rank 2 over Rational Field
```

$$\left(\begin{pmatrix} 0 & 1 \\ 0 & 0 \end{pmatrix}, \begin{pmatrix} 0 & 0 \\ 1 & 0 \end{pmatrix}, \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}\right)$$

False

True

False

El ejemplo más famoso de un álgebra de Lie nilpotente es el *álgebra de Lie de Heisenberg*, la cual tiene una base X, Y, Z con el único corchete no trivial

$$[X, Y] = Z.$$

[89]:
```
h = lie_algebras.Heisenberg(QQ, 1, representation="matrix");
print(h)
display(h.dimension())
X, Y, Z =  h.basis()
display((X, Y, Z))
display(h.gens())
display(X.bracket(Y) == Z)
display(h.is_nilpotent())
display(h.is_semisimple())
```

```
Heisenberg algebra of rank 1 over Rational Field
```

3

$$\left(\begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}, \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{pmatrix}, \begin{pmatrix} 0 & 0 & 1 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}\right)$$

$$\left(\begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}, \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{pmatrix}\right)$$

True

True

False

13. Métricas riemannianas

Una *métrica riemanniana* g, a veces también denotada por $\langle\cdot,\cdot\rangle$, en una variedad diferenciable M asigna a cada $p \in M$ un producto escalar g_p en T_pM de manera diferenciable: en coordenadas locales las funciones

$$g_{ij} = g\left(\frac{\partial}{\partial x_i}, \frac{\partial}{\partial x_j}\right)$$

son funciones diferenciables. Esto permite:

- Calcular la norma de vectores tangentes (y de campos vectoriales): $\|v\| = \sqrt{g(v,v)}$.
- Calcular ángulos entre vectores tangentes $g(v,w) = \|v\|\|w\|\cos\theta$, en donde $v, w \in T_pM$.
- Calcular longitudes de curvas $c : [a,b] \to M$ haciendo

$$L(c) = \int_a^b \|c'(t)\|\, dt.$$

Esto hace de M un espacio métrico y de hecho, la topología métrica inducida por la distancia

$$d(p,q) = \inf_c \{L(c) : c(a) = p, c(b) = q\}$$

coincide con la topología de M.

El ejemplo más sencillo de variedad riemanniana es el espacio euclídeo (en cada punto el producto escalar es el mismo)

[90]:
```
%display plain
E2.<x,y> = EuclideanSpace(2)
E2.categories()
```

[90]:
```
[Join of Category of smooth manifolds over Real Field with 53 bits
   of precision and Category of connected manifolds over Real Field
   with 53 bits of precision and Category of complete metric spaces,
 Category of smooth manifolds over Real Field with 53 bits of
    precision,
 Category of connected manifolds over Real Field with 53 bits of
    precision,
 Category of connected topological spaces,
 Category of differentiable manifolds over Real Field with 53 bits
    of precision,
 Category of manifolds over Real Field with 53 bits of precision,
 Category of complete metric spaces,
 Category of metric spaces,
 Category of topological spaces,
 Category of sets,
 Category of sets with partial maps,
 Category of objects]
```

Sage utiliza la notación tensorial para representar las métricas riemannianas.

[91]:
```
%display latex
display(E2.default_chart())
g = E2.metric()
g.display()
```

$$\left(\mathbb{E}^2, (x,y)\right)$$

[91]:

$$g = \mathrm{d}x \otimes \mathrm{d}x + \mathrm{d}y \otimes \mathrm{d}y$$

En la métrica euclídea los campos coordenados canónicos forman una base ortonormal.

[92]:
```
X, Y = E2.default_frame()
display(g(X, X).display())
display(g(X, Y).display())
display(g(Y, Y).display())
```

$$\begin{array}{rccc} g(e_x, e_x): & \mathbb{E}^2 & \longrightarrow & \mathbb{R} \\ & (x,y) & \longmapsto & 1 \end{array}$$

$$\begin{array}{rccc} g(e_x, e_y): & \mathbb{E}^2 & \longrightarrow & \mathbb{R} \\ & (x,y) & \longmapsto & 0 \end{array}$$

$$\begin{array}{rccc} g(e_y, e_y): & \mathbb{E}^2 & \longrightarrow & \mathbb{R} \\ & (x,y) & \longmapsto & 1 \end{array}$$

También podemos ver la representación matricial de la métrica (por medio de una matriz simétrica definida positiva).

[93]:
```
g[:]
```

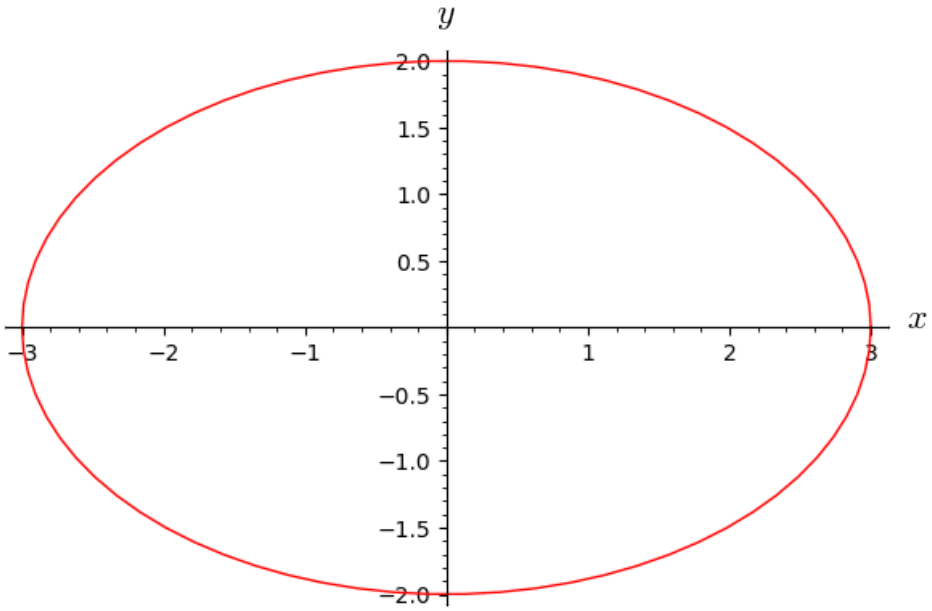
[93]:

$$\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$$

Calculemos a modo de ejemplo la longitud de una elipse (con respecto a la métrica euclídea).

[94]:
```
R.<t> = manifolds.RealLine()
c = E2.curve((3*cos(t),2*sin(t)), (t,0,2*pi), name='c')
c.plot()
```

[94]:



Para ello integramos la norma del vector velocidad.

[95]:
```
v = c.tangent_vector_field()
display(v.display())
print(v.at(R(t)))
display(v.at(R(t)).parent())
v.at(R(t)) in E2.tangent_space(c(t))
```

$$c' = -3\sin(t)\,e_x + 2\cos(t)\,e_y$$

```
Vector c' at Point on the Euclidean plane E^2
```

$$T_{None}\,\mathbb{E}^2$$

[95]:

True

Para calcular la norma de $c'(t)$, necesitamos evaluar la métrica en $c(t)$, es decir usamos el producto interno $g_{c(t)}$ en el espacio tangente $T_{c(t)}\mathbb{E}^2$.

[96]:
```
gt = g.at(c(t))
vt = v.at(R(t))
gt(vt,vt)
```

[96]:

$$4\cos(t)^2 + 9\sin(t)^2$$

Finalmente integramos.

[97]:
```
integrate(sqrt(gt(vt,vt)),t,0,2*pi)
```

[97]:

$$\int_0^{2\pi} \sqrt{4\cos(t)^2 + 9\sin(t)^2}\, dt$$

Si queremos una aproximación numérica podemos usar `.n()`.

[98]:
```
_.n()
```

[98]:

$$15{,}86543958929059$$

13.1. Ejemplo: el plano hiperbólico $\mathbb{H}^2$. Usaremos la presentación del semiplano superior. Es decir, la estructura diferenciable es la usual heredada de $\mathbb{R}^2$,

$$\mathbb{H}^2 = \{(x,y) : y > 0\}$$

y la métrica riemanniana dada en el punto (x,y) por

$$g_{(x,y)} = \frac{1}{y^2}(\mathrm{d}x \otimes \mathrm{d}x + \mathrm{d}y \otimes \mathrm{d}y).$$

[99]:
```
H2 = Manifold(2, name=r'\mathbb H^2', structure='Riemannian')
H2_chart.<x,y> = H2.chart(r'x:(-oo,+oo) y:(0,+oo)')
H2_chart.coord_range()
```

[99]:

$$x\colon (-\infty,+\infty); \quad y\colon (0,+\infty)$$

Introducimos la métrica manualmente:

[100]:
```
g = H2.metric()
g[0,0] = 1/y^2
g[1,1] = 1/y^2
g.display()
```

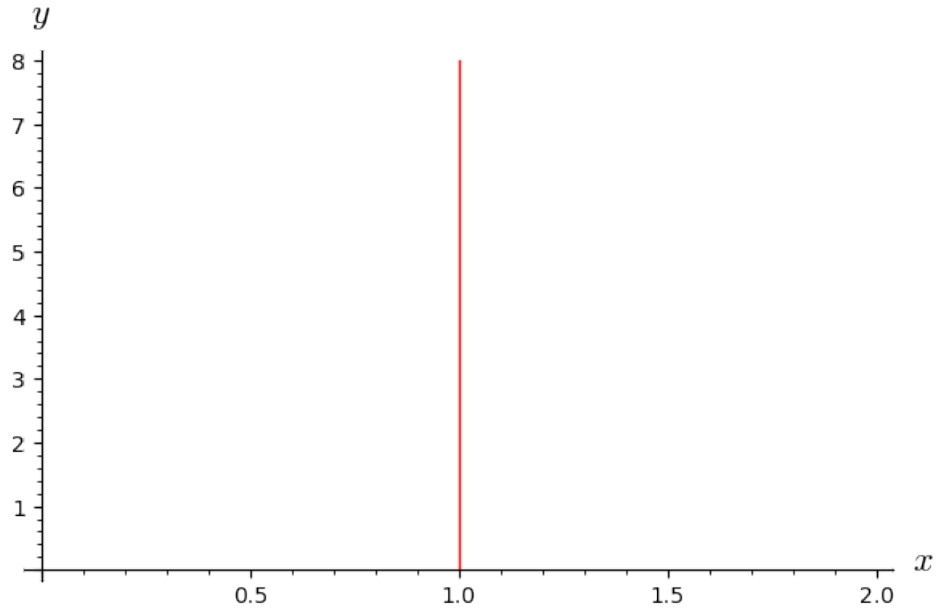
[100]:

$$g = \frac{1}{y^2}\mathrm{d}x \otimes \mathrm{d}x + \frac{1}{y^2}\mathrm{d}y \otimes \mathrm{d}y$$

Consideremos la siguiente curva en $\mathbb{H}^2$:

[101]:
```
R.<t> = manifolds.RealLine()
gamma = H2.curve((1,t), (t,0,oo), name=r"\gamma")
v = gamma.tangent_vector_field()
gamma.plot()
```

[101]:



y calculemos la longitud de la curva $\gamma(t)$, para $t \in [1,8]$, con respecto a la métrica hiperbólica.

[102]:
```
gt = g.at(gamma(t))
vt = v.at(R(t))
integrate(sqrt(gt(vt,vt)),t,1,8)
```

[102]:

$$3\log(2)$$

que aproximadamente es

[103]:
```
_.n()
```

[103]:

$$2{,}07944154167984$$

La longitud euclídea esta curva es claramente 7. La longitud hiperbólica es menor que 7 porque los vectores euclídeos se ven más pequeños con la métrica hiperbólica en los puntos que tienen coordenada $y > 0$. Si calculáramos la longitud de $\gamma(t)$ con $t \in [\frac{1}{2},1]$ nos debería dar más grande que $\frac{1}{2}$.

[104]:
```
display(integrate(sqrt(gt(vt,vt)),t,1/2,1))
_.n()
```

$$\log(2)$$

[104]:

$$2{,}07944154167984$$

14. La conexión de Levi-Civita

Asociada a una métrica riemanniana tenemos una *conexión*

$$\nabla : \mathfrak{X}(M) \times \mathfrak{X}(M) \to \mathfrak{X}(M), \qquad (X,Y) \mapsto \nabla_X Y$$

llamada conexión de Levi-Civita caracterizada por las propiedades:

- ∇ es $\mathbb{R}$ bilineal;
- ∇ es tensorial en la primera variable: $\nabla_{fX}Y = f\nabla_X Y$, para toda $f \in C^\infty(M)$;
- se satisface la regla de Leibniz: $\nabla_X(fY) = X(f)Y + f\nabla_X Y$, para toda $f \in C^\infty(M)$;
- ∇ es una conexión métrica: $X(g(Y,Z)) = g(\nabla_X Y, Z) + g(Y, \nabla_X Z)$ * ∇ es sin torsión: $\nabla_X Y - \nabla_Y X = [X,Y]$.

Remarcamos muy rápidamente algunas similitudes y diferencias entre entre el corchete de Lie y la conexión de Levi-Civita

- Ambos se generalizan para poder derivar objetos más generales que campos (tensores). La generalización del corchete de Lie se llama derivada de Lie y suele denotarse por $\mathscr{L}$, en tanto que la generalización de la conexión de Levi-Civita se denota por el mismo símbolo.
- Sin embargo, una detecta simetrías y la otra paralelismo. Más precisamente, si X es un campo vectorial y T es un tensor,
  - $\mathscr{L}_X T = 0$ dice que el flujo de X preserva T.
  - $\nabla_X T = 0$ dice que T es paralelo en la dirección de X.
- El corchete/derivada de Lie es independiente de la métrica.
- La conexión de Levi-Civita está unívocamente determinada por la métrica.
- La conexión de Levi-Civita permite derivar objetos que estén definidos solo a lo largo de curvas. En particular, nos permite definir la aceleración de una curva (esto no se puede hacer con la derivada de Lie).

14.1. Símbolos de Christoffel. Si M es una variedad riemanniana y $x_1, \ldots, x_n$ es una carta local, las propiedades anteriores nos dicen que para calcular $\nabla_X Y$, es suficiente con saber calcular $\nabla_{\frac{\partial}{\partial x_i}} \frac{\partial}{\partial x_j}$. Estos campos quedan determinados por los *símbolos de Christoffel* Γ_{ij}^k (en la carta dada)

$$\nabla_{\frac{\partial}{\partial x_i}} \frac{\partial}{\partial x_j} = \sum_k \Gamma_{ij}^k \frac{\partial}{\partial x_k}.$$

Sage nos permite calcular la conexión de Levi-Civita con `g.connection()` y nos muestra los símbolos de Christoffel con `g.connection().display()`. Por ejemplo para el plano hiperbólico:

[105]:
```
nabla = g.connection()
nabla.display()
```

[105]:

$$\begin{aligned} \Gamma^{x}{}_{xy} &= -\frac{1}{y} \\ \Gamma^{x}{}_{yx} &= -\frac{1}{y} \\ \Gamma^{y}{}_{xx} &= \frac{1}{y} \\ \Gamma^{y}{}_{yy} &= -\frac{1}{y} \end{aligned}$$

Por defecto solo se muestran los símbolos de Christoffel no nulos, pero podemos pedir que se muestren todos.

[106]:
```
nabla.display(only_nonzero=False)
```

[106]:

$$\begin{aligned}
\Gamma^{x}{}_{xx} &= 0\\
\Gamma^{x}{}_{xy} &= -\tfrac{1}{y}\\
\Gamma^{x}{}_{yx} &= -\tfrac{1}{y}\\
\Gamma^{x}{}_{yy} &= 0\\
\Gamma^{y}{}_{xx} &= \tfrac{1}{y}\\
\Gamma^{y}{}_{xy} &= 0\\
\Gamma^{y}{}_{yx} &= 0\\
\Gamma^{y}{}_{yy} &= -\tfrac{1}{y}
\end{aligned}$$

También podemos acceder a los símbolos de Christoffel usando índices, notar que Γ^k_{ij} se calcula con `nabla[i,j,k]`.

[107]:
```
display(nabla[1,0,0])
display(nabla[1,0,1])
display(nabla[1,1,1])
```

$$\frac{1}{y}$$

$$0$$

$$-\frac{1}{y}$$

15. Geodésicas

Una curva $\gamma(t)$ en una variedad riemanniana M se dice una *geodésica* si su velocidad es paralela (a veces también se dice que tiene aceleración nula, pero esto se puede prestar a confusión)

$$\nabla_{\gamma'(t)}\gamma'(t) = 0.$$

SageManifolds todavía no tiene implementada la derivada covariante a lo largo de curvas, pero podemos ayudarnos de los símbolos de Christoffel para estudiar la existencia de geodésicas.

15.1. La ecuación geodésica. Si $\gamma(t)$ es una geodésica tal que $\gamma(0) = p$ y $\gamma'(0) = v \in T_pM$, se satisface

$$\gamma_k'' + \sum_{i,j} \Gamma^k_{ij}\gamma_i'\gamma_j' = 0$$

En particular, siempre existen las geodésicas (para t pequeño) en cualquier dirección y las geodésicas son las curvas que localmente minimizan la distancia.

15.2. Geodésicas en el plano hiperbólico. Se puede probar que las geodésicas en nuestra versión del plano hiperbólico son las semirrectas paralelas al eje y y los semicírculos que intersectan perpendicularmente el eje x. El siguiente ejemplo fue tomado del foro de discusión de Sage y puede consultarse en https://ask.sagemath.org/question/48915/check-if-a-curve-is-a-geodesic/.

[108]:
```
H2 = Manifold(2, r'\mathbb H^2', structure='Riemannian')
X.<x,y> = H2.chart(r'x y:(0,+oo)')
g = H2.metric()
```

```
g[0, 0], g[1, 1] = 1/y^2, 1/y^2
g.display()
```

[108]:

$$g = \frac{1}{y^2}\mathrm{d}x \otimes \mathrm{d}x + \frac{1}{y^2}\mathrm{d}y \otimes \mathrm{d}y$$

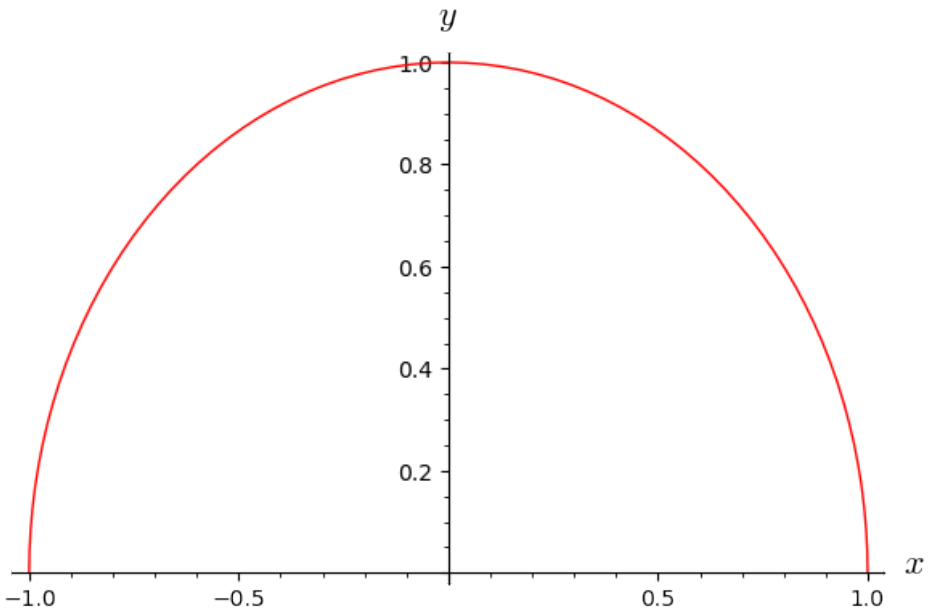
Definamos la siguiente curva

[109]:
```
R.<t> = manifolds.RealLine()
gamma = H2.curve([cos(t), sin(t)], (t, 0, pi), name=r'\gamma')
display(gamma.display())
v = gamma.tangent_vector_field()
display(v.display())
gamma.plot()
```

$$\begin{array}{rccc} \gamma\colon & (0,\pi) & \longrightarrow & \mathbb{H}^2 \\ & t & \longmapsto & (x,y) = (\cos(t), \sin(t)) \end{array}$$

$$\gamma' = -\sin(t)\frac{\partial}{\partial x} + \cos(t)\frac{\partial}{\partial y}$$

[109]:



Nos ayudamos de los símbolos de Christoffel para estudiar la ecuación geodésica.

[110]:
```
cc = g.christoffel_symbols()
a = gamma.domain().vector_field(dest_map=gamma)
for i in H2.irange():
    a[i] = diff(v[i].expr(), t) \
           + sum(sum(cc[i, j, k](*X(gamma(t)))*v[j]\
                     .expr()*v[k].expr()
```

```
                    for j in H2.irange()) for k in H2.irange())

print(a)
a.display()
```

```
Vector field along the Real interval (0, pi) with values on the
  2-dimensional Riemannian manifold \mathbb H^2
```

[110]:

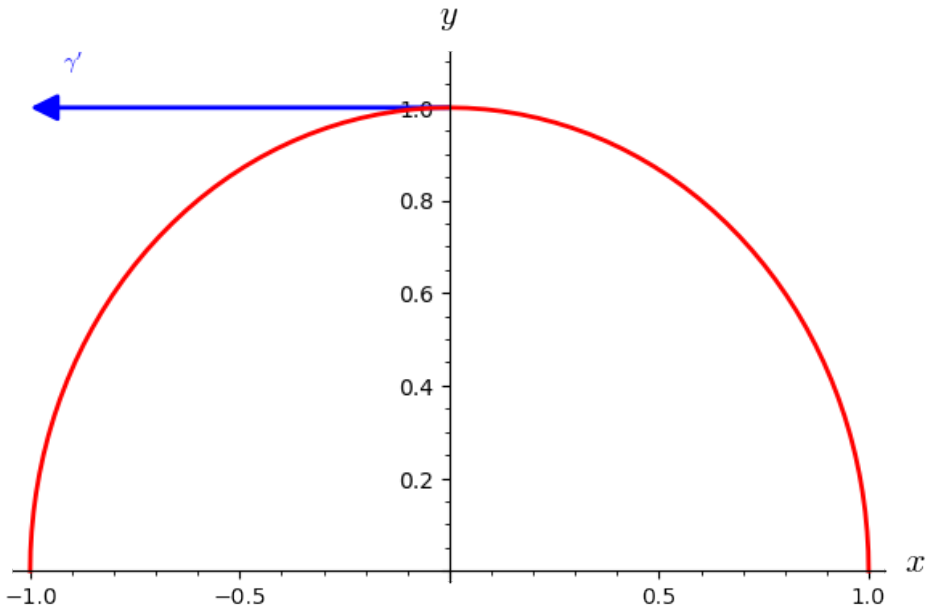
$$\cos(t)\frac{\partial}{\partial x} - \frac{\cos(t)^2}{\sin(t)}\frac{\partial}{\partial y}$$

Lo anterior nos dice que $\gamma(t)$ no es una geodésica. Sin embargo, sabemos que la imagen de γ es la imagen de una geodésica en el plano hiperbólico (lo que falla entonces es la parametrización). Tratemos de encontrar cuál es esa geodésica. Primero elijamos las condiciones iniciales.

[111]:
```
display(gamma(pi/2).coord())
v.at(R(pi/2)).plot() + gamma.plot(thickness=2)
```

$$(0,1)$$

[111]:



A continuación definimos una curva genérica $\gamma(t)$ con coordenadas $f_1(t), f_2(t)$. Las condiciones iniciales deberían ser:

$$f_1(0) = 1 \qquad f_1'(0) = 0$$
$$f_2(0) = 1 \qquad f_2'(0) = -1$$

[112]:
```
f1 = function('f_1')(t)
f2 = function('f_2')(t)
```

Lo que estamos haciendo con esto es definir expresiones simbólicas que dependen de `t`.

[113]:
```
print(type(f1))
```

```
<class 'sage.symbolic.expression.Expression'>
```

Las cuales luego pueden ser usadas para definir una curva genérica.

[114]:
```
gamma = H2.curve([f1(t), f2(t)], t, name=r'\gamma')
v = gamma.tangent_vector_field()
v.display()
```

```
/tmp/ipykernel_25603/880341076.py:1: DeprecationWarning: Substitution
  using function-call syntax and unnamed arguments is deprecated and
  will be removed from a future release of Sage; you can use named
  arguments instead, like EXPR(x=..., y=...)
See http://trac.sagemath.org/5930 for details.
  gamma = H2.curve([f1(t), f2(t)], t, name=r'\gamma')
```

[114]:

$$\gamma = \frac{\partial f_1}{\partial t}\frac{\partial}{\partial x} + \frac{\partial f_2}{\partial t}\frac{\partial}{\partial y}$$

Aquí recibimos una amable advertencia de que el método que estamos usando para definir γ es ahora obsoleto.

[115]:
```
cc = g.christoffel_symbols()
a = gamma.domain().vector_field(dest_map=gamma)
for i in H2.irange():
    a[i] = diff(v[i].expr(), t) \
           + sum(sum(cc[i, j, k](*X(gamma(t)))*v[j]\
                     .expr()*v[k].expr()
                     for j in H2.irange()) for k in H2.irange())

print(a)
a.display()
```

```
Vector field along the Real number line R with values on the
  2-dimensional Riemannian manifold \mathbb H^2
```

[115]:

$$\left(-\frac{2\frac{\partial f_1}{\partial t}\frac{\partial f_2}{\partial t}}{f_2(t)} + \frac{\partial^2 f_1}{\partial t^2}\right)\frac{\partial}{\partial x} + \left(\frac{\left(\frac{\partial f_1}{\partial t}\right)^2}{f_2(t)} - \frac{\left(\frac{\partial f_2}{\partial t}\right)^2}{f_2(t)} + \frac{\partial^2 f_2}{\partial t^2}\right)\frac{\partial}{\partial y}$$

Habría que resolver estas ecuaciones diferenciales, pero lo vamos a dejar acá.

16. Isometrías

Una *isometría* entre dos variedades riemannianas es un difeomorfismo $f : M \to N$ que preserva la métrica. Es decir,

$$\mathrm{d}f|_p : T_pM \to T_{f(p)}N$$

es una isometría lineal para cada $p \in M$. Se tiene que

$$I(M) = \{f : M \to M : f \text{ es una isometría}\}$$

forma un grupo con la composición, llamado el *grupo de isometrías* de M. El grupo $I(M)$ tiene una topología natural llamada topología compacto abierta y es bien conocido el siguiente resultado (ver por ejemplo [KN69]).

Teorema. *$I(M)$ es un grupo de Lie (con la topología compacto abierta) y si $\dim M = n$, entonces*

$$\dim I(M) \leq \frac{n(n+1)}{2}.$$

Si se da la igualdad en el teorema anterior, entonces M resulta ser de una clase muy especial: espacios de curvatura constante (veremos algo de esto en breve).

El álgebra de Lie del grupo de isometrías está dada por los campos de Killing: un campo $X \in \mathfrak{X}(M)$ se dice un *campo de Killing* si ∇X es antisimétrica con respecto a la métrica, es decir:

$$\langle \nabla_Y X, Z\rangle + \langle Y, \nabla_Z X\rangle = 0$$

para todos $Y, Z \in \mathfrak{X}(M)$. Hay una criterio muy sencillo (en teoría) para verificar si un campo de X es un campo de Killing.

Teorema. *X es un campo de Killing si y sólo si la derivada de Lie de la métrica en la dirección de X es cero:*

$$\mathscr{L}_X \langle \cdot, \cdot \rangle = 0.$$

Llevaría un buen tiempo definir qué es la derivada de Lie (es la extensión de la que hablamos antes del corchete de Lie para poder derivar todo tipo de tensores). Afortunadamente, Sage sabe calcular derivadas de Lie. Sigamos explorando el ejemplo del plano hiperbólico

[116]:
```
H2 = Manifold(2, r'\mathbb H^2', structure='Riemannian')
H2_chart.<x,y> = H2.chart(r'x y:(0,+oo)')
g = H2.metric()
g[0, 0], g[1, 1] = 1/y^2, 1/y^2
nabla = g.connection()
```

Calculemos las derivadas de Lie de la métrica en las direcciones de los campos coordenados.

[117]:
```
X, Y = H2.default_frame()
display(g.lie_derivative(X).disp())
display(g.lie_derivative(Y).disp())
```

$$0$$

$$-\frac{2}{y^3}\mathrm{d}x\otimes\mathrm{d}x - \frac{2}{y^3}\mathrm{d}y\otimes\mathrm{d}y$$

De lo anterior sigue que $X = \partial/\partial x$ es un campo de Killing. Verifiquemos que efectivamente ∇X es un tensor antisimétrico

[118]:
```
display(nabla(X))
nabla(X).display()
```

$$\nabla_g \frac{\partial}{\partial x}$$

[118]:

$$\nabla_g \frac{\partial}{\partial x} = -\frac{1}{y}\frac{\partial}{\partial x} \otimes \mathrm{d}y + \frac{1}{y}\frac{\partial}{\partial y} \otimes \mathrm{d}x$$

Otra forma de hacer esto es observando que la matriz de ∇X es antisimétrica, con respecto al producto interno asociado a la métrica riemanniana (que en este caso es un múltiplo, que varía punto a punto, del producto interno usual).

[119]:
```
g[:], nabla(X)[:], nabla(Y)[:]
```

[119]:

$$\left(\begin{pmatrix} \frac{1}{y^2} & 0 \\ 0 & \frac{1}{y^2} \end{pmatrix}, \begin{pmatrix} 0 & -\frac{1}{y} \\ \frac{1}{y} & 0 \end{pmatrix}, \begin{pmatrix} -\frac{1}{y} & 0 \\ 0 & -\frac{1}{y} \end{pmatrix}\right)$$

17. Curvatura

El *tensor de curvatura* es uno de los objetos más importantes asociados a una variedad riemanniana M, el cual captura casi toda la información geométrica de M. Usualmente denotado por R, es un tensor de tipo $(1,3)$, lo cual informalmente podemos pensar como una aplicación $C^\infty(M)$-multilineal: recibe tres campos y devuelve un campo

$$R : \mathfrak{X}(M) \times \mathfrak{X}(M) \times \mathfrak{X}(M) \to \mathfrak{X}(M)$$

y se define como

$$R_{X,Y}Z = \nabla_X \nabla_Y Z - \nabla_Y \nabla_X Z - \nabla_{[X,Y]} Z$$

En Sage podemos acceder al tensor de curvatura de la métrica g usando el método `g.riemann()`.

[120]:
```
Riem = g.riemann()
print(Riem)
Riem.parent()
```

```
Tensor field Riem(g) of type (1,3) on the 2-dimensional Riemannian
  manifold \mathbb H^2
```

[120]:

$$\mathscr{T}^{(1,3)}\left(\mathbb{H}^2\right)$$

En coordenadas locales, el tensor de curvatura tiene el siguiente aspecto.

[121]:
```
%display plain
Riem.display()
```

[121]:
```
Riem(g) = -1/y^2 ∂/∂x⊗dy⊗dx⊗dy + y^(-2) ∂/∂x⊗dy⊗dy⊗dx
  + y^(-2) ∂/∂y⊗dx⊗dx⊗dy - 1/y^2 ∂/∂y⊗dx⊗dy⊗dx
```

A veces conviene pensar al tensor de curvatura como un tensor de tipo $(0,4)$, es decir, recibe 4 campos y devuelve una función. Esto se hace contrayendo con la métrica

$$(X,Y,Z,W) \mapsto \langle R_{X,Y}Z, W\rangle$$

Sage sabe hacer esto con el método `.down(g)`, pero hay que tener cuidado con los signos (ya que no hay una convención universal sobre el orden en el cual ponemos las variables del tensor de curvatura).

[122]:
```
%display latex
Riem1 = Riem.down(g)
print(Riem1)
```

```
Tensor field of type (0,4) on the 2-dimensional Riemannian manifold
  \mathbb H^2
```

[123]:
```
Riem1.display()
```

[123]:

$$-\frac{1}{y^4}\mathrm{d}x\otimes\mathrm{d}y\otimes\mathrm{d}x\otimes\mathrm{d}y+\frac{1}{y^4}\mathrm{d}x\otimes\mathrm{d}y\otimes\mathrm{d}y\otimes\mathrm{d}x+\frac{1}{y^4}\mathrm{d}y\otimes\mathrm{d}x\otimes\mathrm{d}x\otimes\mathrm{d}y-\frac{1}{y^4}\mathrm{d}y\otimes\mathrm{d}x\otimes\mathrm{d}y\otimes\mathrm{d}x$$

Verifiquemos que $\langle R,\cdot\rangle$ se puede obtener con `Riem.down(g)`. En primer lugar definamos cuatro campos genéricos.

[124]:
```
a0,a1,b0,b1,c0,c1,d0,d1 = var("a0,a1,b0,b1,c0,c1,d0,d1")
X1 = a0*X + a1*Y
X2 = b0*X + b1*Y
X3 = c0*X + c1*Y
X4 = d0*X + d1*Y
X1.display(), X2.display(), X3.display(), X4.display()
```

[124]:

$$\left(a_0\frac{\partial}{\partial x}+a_1\frac{\partial}{\partial y}, b_0\frac{\partial}{\partial x}+b_1\frac{\partial}{\partial y}, c_0\frac{\partial}{\partial x}+c_1\frac{\partial}{\partial y}, d_0\frac{\partial}{\partial x}+d_1\frac{\partial}{\partial y}\right)$$

Observemos los siguiente, cuando decimos que R es un tensor, estamos pensando en que es $C^\infty(M)$-multilineal (i.e., saca funciones diferenciables multiplicando). Por ende para calcular $R_{X,Y}Z$ en un punto p, solo necesitamos conocer X_p, Y_p, Z_p. En ese sentido es que los campos X_1, X_2, X_3, X_4 que definimos más arriba son genéricos, ya que son combinaciones lineales (sobre $\mathbb{R}$) de un frame global.

Calculamos por un lado

[125]:
```
comp_1 = Riem1(X1,X2,X3,X4).expr().expand()
comp_1
```

[125]:

$$-\frac{a_1b_0c_1d_0}{y^4}+\frac{a_0b_1c_1d_0}{y^4}+\frac{a_1b_0c_0d_1}{y^4}-\frac{a_0b_1c_0d_1}{y^4}$$

y por el otro

[126]:
```
comp_2 = g(nabla(nabla(X3)(X2))(X1) \
           - nabla(nabla(X3)(X1))(X2), X4).expr().expand()
comp_2
```

[126]:

$$\frac{a_1b_0c_1d_0}{y^4} - \frac{a_0b_1c_1d_0}{y^4} - \frac{a_1b_0c_0d_1}{y^4} + \frac{a_0b_1c_0d_1}{y^4}$$

y verificamos que una es menos la otra.

[127]:
```
comp_1 + comp_2
```

[127]:

$$0$$

18. Curvatura seccional

La curvatura seccional es la versión de la curvatura gaussiana para variedades n dimensionales. De hecho, cuando $n = 2$, nos dan la misma información. La curvatura secciona K en un punto p de M es un valor que se asigna a cada subespacio bidimensional de T_pM. Si u, v es una base de $\mathbb{V} \subset T_pM$

$$K(\mathbb{V}) = K(u,v) = \frac{\langle R_{u,v}v, u\rangle}{\langle u,u\rangle\langle v,v\rangle - \langle u,v\rangle^2}.$$

Se puede probar que está definición es independiente de la base elegida. Si $n \geq 3$, K brinda menos información que el tensor de curvatura, pero es un invariante más sencillo.

La curvatura seccional no viene por defecto en Sage, hay que definirla. Verifiquemos que el plano hiperbólico tiene curvatura constante -1.

[128]:
```
(Riem1(X1,X2,X1,X2) / (g(X1,X1) * g(X2,X2) - g(X1,X2)^2)).expr()
```

[128]:

$$-1$$

Y ya que estamos, definamos una función que calcule la curvatura seccional.

[129]:
```
def curv_secc(u,v):
    p = u.parent().base_point()
    M = p.parent()
    g = M.metric()
    gp = g.at(p)
    Riem = g.riemann().down(g).at(p)
    return (Riem(u,v,u,v) / \
            (gp(u,u)*gp(v,v) - gp(u,v)^2)).simplify_full()
p = H2.point(H2_chart[:])
u, v = H2_chart.frame().at(p)
curv_secc(u, v)
```

[129]:

$$-1$$

19. Espacios de curvatura constante

Los espacios de curvatura constantes son aquellos en los que la curvatura seccional es independiente del punto (y del plano elegido en el espacio). Hay 3 modelos simplemente conexos de espacios de curvatura constante en cada dimensión $n \geq 2$:

- $\mathbb{R}^n$ con la métrica euclídea es un espacio de curvatura constante nula (también llamado plano o flat);

- $\mathbb{S}^n$ con la métrica redonda (heredada de $\mathbb{R}^{n+1}$) es un espacio de curvatura constante positiva;
- $\mathbb{H}^n$ con la métrica hiperbólica es un espacio de curvatura constante negativa.

Para terminar el curso tratemos de probar el siguiente resultado con lo que aprendimos hasta ahora.

Teorema. *Sea M una variedad riemanniana simplemente conexa de dimensión 3 y curvatura constante. Entonces existe un grupo de Lie G con una métrica invariante a izquierda isométrico a M.*

19.1. Ejemplo: flat (con grupo de Lie abeliano). Es fácil ver a mano que el espacio euclídeo $\mathbb{R}^3$ tiene curvatura nula. Más aún, probemos que cualquier métrica constante en $\mathbb{R}^3$ tiene curvatura cero (notar que $\mathbb{R}^3$ es un grupo de Lie abeliano y los campos coordenados usuales son invariantes a izquierda).

[130]:
```
M = Manifold(3, r'\mathbb R^3', structure='Riemannian')
M_chart.<x,y,z> = M.chart()
M_frame = M_chart.frame()
g = M.metric()
```

Definimos las siguientes variables simbólicas para construir una métrica constante arbitraria:

[131]:
```
mu, nu, eta, alpha, beta, gamma \
= var("mu, nu, eta, alpha, beta, gamma")
g[0,0], g[1,1], g[2,2], g[0,1], g[0,2], g[1,2] \
= mu, nu, eta, alpha, beta, gamma
g[:] #esta matriz debe ser simétrica y definida positiva
```

[131]:

$$\begin{pmatrix} \mu & \alpha & \beta \\ \alpha & \nu & \gamma \\ \beta & \gamma & \eta \end{pmatrix}$$

Veamos que los campos coordenados son invariantes a izquierda.

[132]:
```
x0, y0, z0 = var('x0 y0 z0')
p = M.point((x0, y0, z0))
Lp = M.diffeomorphism(M, (x + x0, y + y0, z + z0), name='L_p')
Lp.display()
```

[132]:

$$\begin{array}{rccc} L_p: & \mathbb{R}^3 & \longrightarrow & \mathbb{R}^3 \\ & (x,y,z) & \longmapsto & (x+x_0, y+y_0, z+z_0) \end{array}$$

Aquí L_p es la traslación a izquierda en $p = (x_0, y_0, z_0)$ y su diferencial se identifica con la transformación identidad.

[133]:
```
zero = M.point((0, 0, 0))
dLp_0 = Lp.differential(zero)
dLp_0.matrix()
```

[133]:

$$\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

Como los campos coordenados tienen coordenadas constantes en el frame `M_frame`, sigue que $\mathrm{d}L_p$ preserva los campos invariantes a izquierda. Además, como la métrica `g` tiene coeficientes constantes en un frame invariante a izquierda, resulta una métrica invariante a izquierda.

Por último, verifiquemos que la métrica es plana.

[134]:
```
Riem = g.riemann()
%time Riem.display()
```

```
CPU times: user 1.17 ms, sys: 0 ns, total: 1.17 ms
Wall time: 1.19 ms
```

[134]:

$$\mathrm{Riem}(g) = 0$$

19.2. Ejemplo: flat (con grupo de Lie soluble no abeliano). Es un poco más interesante observar que existe un grupo de Lie no abeliano que admite una métrica plana. Consideremos el grupo euclídeo $E(2)$ de movimientos rígidos del plano. Se puede ver que:

- $E(2)$ es un grupo de Lie de dimensión 3 y su estructura de grupo queda determinado por el siguiente hecho: todo movimiento rígido es composición de una traslación T_v, $v \in \mathbb{R}^2$ y una rotación R_t, alrededor del origen, en un ángulo $t \in \mathbb{R}$:

$$\begin{aligned}(T_v \circ R_t) \circ (T_w \circ R_s) &= (T_v \circ R_t \circ T_w \circ (R_t)^{-1}) \circ (R_t \circ R_s) \\ &= (T_v \circ T_{R_t(w)}) \circ R_{t+s} \\ &= T_{v+R_t(w)} \circ R_{t+s}.\end{aligned}$$

- En coordenadas $(v,t) = (x,y,t)$ el producto en el cubrimiento universal de $E(2)$ queda

$$(x,y,t)(x',y',t') = (x + x'\cos t - y'\sin t, y + x'\sin t + y'\cos t, t + t').$$

[135]:
```
M = Manifold(3, r'\tilde E(2)',structure='Riemannian')
M_chart.<x,y,t> = M.chart()
g = M.metric()
```

Definimos las traslaciones a izquierda usando la fórmula anterior

[136]:
```
x0, y0, t0 = var("x0 y0 t0")
p = M.point((x0, y0, t0))
Lp = M.diffeomorphism(M, \
                      (x0+x*cos(t0)-y*sin(t0), \
                       y0+x*sin(t0)+y*cos(t0),t0+t), \
                      name='L_p')
Lp.display()
```

[136]:

$$\begin{aligned} L_p : \quad \tilde{E}(2) &\longrightarrow \tilde{E}(2) \\ (x,y,t) &\longmapsto (x\cos(t_0) - y\sin(t_0) + x_0, y\cos(t_0) + x\sin(t_0) + y_0, t + t_0)\end{aligned}$$

y calculamos su diferencial

[137]:
```
zero = M.point((0,0,0))
dLp_0 = Lp.differential(zero)
dLp_0.matrix()
```

[137]:

$$\begin{pmatrix} \cos(t_0) & -\sin(t_0) & 0 \\ \sin(t_0) & \cos(t_0) & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

Las columnas de la matriz `dLp_0.matrix()` nos dan los coeficientes del frame invariante asociado, con respecto a `M.default_frame()`.

[138]:
```
X, Y, T = M.default_frame()
L0 = M.vector_field([cos(t),sin(t),0], name='L_0')
L1 = M.vector_field([-sin(t),cos(t),0], name='L_1')
L2 = M.vector_field([0,0,1], name='L_2')
M_left_inv_frame = M.vector_frame('L',(L0,L1,L2))
M.frames()
```

[138]:

$$\left[\left(\tilde{E}(2), \left(\frac{\partial}{\partial x}, \frac{\partial}{\partial y}, \frac{\partial}{\partial t}\right)\right), \left(\tilde{E}(2), (L_0, L_1, L_2)\right)\right]$$

Verifiquemos que el corchete de campos invariantes a izquierda es invariante a izquierda.

[139]:
```
display(L0.bracket(L1).display(M_left_inv_frame))
display(L0.bracket(L2).display(M_left_inv_frame))
display(L1.bracket(L2).display(M_left_inv_frame))
```

$$[L_0, L_1] = 0$$

$$[L_0, L_2] = -L_1$$

$$[L_1, L_2] = L_0$$

Para definir una métrica invariante a izquierda hay que usar un pequeño truco ya que a `.metric()` no podemos pasar el el parámetro `frame=` pero a `.tensor_field()` sí. Entonces creamos un tensor `g_aux` con los mismo coeficientes que queremos para nuestra métrica en un frame invariante a izquierda y luego definimos los coeficientes de `g[i,j] = g_aux[i,j]` en el frame `M.default_frame()`.

[140]:
```
mu, nu, eta = var("mu, nu, eta")
assume(mu>0,nu>0,eta>0)
g_aux = M.tensor_field(0,2,\
                       [[mu,0,0],[0,nu,0],[0,0,eta]],\
                       frame=M_left_inv_frame,name='g_a')
g[:] = g_aux[:]
%display plain
g.display()
```

[140]:
```
g = ((mu - nu)*cos(t)^2 + nu) dx⊗dx + (mu - nu)*cos(t)*sin(t) dx⊗dy
  + (mu - nu)*cos(t)*sin(t) dy⊗dx
  + (-(mu - nu)*cos(t)^2 + mu) dy⊗dy + eta dt⊗dt
```

La métrica es invariante a izquierda ya que tiene coeficientes constantes con respecto al frame invariante a izquierda.

[141]:
```
%display latex
%time g.display(M_left_inv_frame)
```

```
CPU times: user 1.08 s, sys: 9.68 ms, total: 1.09 s
Wall time: 900 ms
```

[141]:

$$g = \mu L^0 \otimes L^0 + \nu L^1 \otimes L^1 + \eta L^2 \otimes L^2$$

Definimos el tensor de curvatura `Riem` de la métrica `g`.

[142]:
```
%time Riem = g.riemann() #long time
```

```
CPU times: user 7.22 s, sys: 86.9 ms, total: 7.31 s
Wall time: 5.91 s
```

Hacer un `Riem.display(M_left_inv_frame)` lleva mucho tiempo, podemos acelerar un poco la cosa con `Parellelism()`.

[143]:
```
Parallelism().set(nproc=8)
%display plain
%time Riem.display(M_left_inv_frame) #long time
```

```
CPU times: user 447 ms, sys: 114 ms, total: 561 ms
Wall time: 16.5 s
```

[143]:
```
Riem(g) = 1/4*(mu^2 - 2*mu*nu + nu^2)/(eta*mu) L_0⊗L^1⊗L^0⊗L^1
  - 1/4*(mu^2 - 2*mu*nu + nu^2)/(eta*mu) L_0⊗L^1⊗L^1⊗L^0
  + 1/4*(mu^2 + 2*mu*nu - 3*nu^2)/(mu*nu) L_0⊗L^2⊗L^0⊗L^2
  - 1/4*(mu^2 + 2*mu*nu - 3*nu^2)/(mu*nu) L_0⊗L^2⊗L^2⊗L^0
  - 1/4*(mu^2 - 2*mu*nu + nu^2)/(eta*nu) L_1⊗L^0⊗L^0⊗L^1
  + 1/4*(mu^2 - 2*mu*nu + nu^2)/(eta*nu) L_1⊗L^0⊗L^1⊗L^0
  - 1/4*(3*mu^2 - 2*mu*nu - nu^2)/(mu*nu) L_1⊗L^2⊗L^1⊗L^2
  + 1/4*(3*mu^2 - 2*mu*nu - nu^2)/(mu*nu) L_1⊗L^2⊗L^2⊗L^1
  - 1/4*(mu^2 + 2*mu*nu - 3*nu^2)/(eta*nu) L_2⊗L^0⊗L^0⊗L^2
  + 1/4*(mu^2 + 2*mu*nu - 3*nu^2)/(eta*nu) L_2⊗L^0⊗L^2⊗L^0
  + 1/4*(3*mu^2 - 2*mu*nu - nu^2)/(eta*mu) L_2⊗L^1⊗L^1⊗L^2
  - 1/4*(3*mu^2 - 2*mu*nu - nu^2)/(eta*mu) L_2⊗L^1⊗L^2⊗L^1
```

Otra opción sería ir pidiéndole a Sage que haga cálculos más sencillos. Por ejemplo,

[144]:
```
%display latex
Riem[1,0,1,0].factor()
```

[144]:

$$\frac{\left(\mu\cos(t)^2-\nu\cos(t)^2+\nu\right)(\mu-\nu)^2}{4\,\eta\,\mu\,\nu}$$

nos dice que para que la métrica sea plana necesitamos que $\mu=\nu$. Y podemos empezar de nuevo...

[145]:
```
g_aux = M.tensor_field(0,2,\
                       [[mu,0,0],[0,mu,0],[0,0,eta]],\
                       frame=M_left_inv_frame,name='g_a')
g[:] = g_aux[:]
%time Riem = g.riemann() #tarda mucho menos
```

```
CPU times: user 176 ms, sys: 295 ms, total: 471 ms
Wall time: 873 ms
```

Con esto ya hacemos que la métrica sea plana:

[146]:
```
Riem.display()
```

[146]:

$$\mathrm{Riem}(g)=0$$

Esto también lo podemos comprobar viendo que la métrica invariante a izquierda coincide con una métrica euclídea.

[147]:
```
g.display()
g.display(M_left_inv_frame)
```

[147]:

$$g=\mu L^0\otimes L^0+\mu L^1\otimes L^1+\eta L^2\otimes L^2$$

19.3. Ejemplo: curvatura negativa. Para estudiar el caso del espacio hiperbólico nos ayudaremos de ciertos resultados que son conocidos en la literatura. Consideremos el grupo de Lie soluble G, cuya álgebra de Lie $\mathfrak{g}$ admite una base e_0,e_1,e_2 con los corchetes dados por

$$[L_0,L_1]=0,\qquad [L_2,L_0]=L_0\qquad [L_2,L_1]=L_1$$

Se puede ver que el grupo de Lie simplemente conexo asociado a esta álgebra de Lie es $\mathbb{R}^3$ con el producto

$$(x,y,t)(x',y',t')=(x+e^t x',y+e^t y',t+t').$$

Sabiendo esto, ya podemos implementarlo en Sage.

[148]:
```
M = Manifold(3, 'M',structure='Riemannian')
M_chart.<x,y,t> = M.chart()
g = M.metric()
x0, y0, t0 = var("x0 y0 t0")
p = M.point((x0, y0, t0), name='p')
zero = M.point((0,0,0))
Lp = M.diffeomorphism(M, (x0+x*exp(t0),y0+y*exp(t0),t0+t), \
                      name='L_p')
Lp.display()
```

[148]:

$$\begin{array}{rccc} L_p: & M & \longrightarrow & M \\ & (x,y,t) & \longmapsto & (xe^{t_0}+x_0, ye^{t_0}+y_0, t+t_0) \end{array}$$

Ejercicio. Verificar que M es un grupo de Lie. *Ayuda*: Probar que el producto es asociativo definiendo otro punto arbitrario `q`, los difeomorfismos `Lq`, `Lpq` y chequeando que `Lp.post_compose(Lq) == Lpq`. Finalmente, probar que existen inversos verificando que `Lp.inverse() == Lq` con `q = Lp.inverse()(zero)`.

Para calcular los campos invariantes a izquierda usamos el mismo truco que antes.

[149]:
```
dLp_0 = Lp.differential(zero)
dLp_0.matrix()
```

[149]:

$$\begin{pmatrix} e^{t_0} & 0 & 0 \\ 0 & e^{t_0} & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

Y como antes definimos un frame invariante a izquierda.

[150]:
```
X, Y, T = M.default_frame()
L0 = M.vector_field([exp(t),0,0], name='L_0')
L1 = M.vector_field([0,exp(t),0], name='L_1')
L2 = M.vector_field([0,0,1], name='L_2')
M_left_inv_frame = M.vector_frame('L',(L0,L1,L2))
M.frames()
```

[150]:

$$\left[\left(M, \left(\frac{\partial}{\partial x}, \frac{\partial}{\partial y}, \frac{\partial}{\partial t}\right)\right), (M, (L_0, L_1, L_2))\right]$$

Chequeemos el corchete de Lie satisface las relaciones propuestas

[151]:
```
display(L0.bracket(L1).display(M_left_inv_frame))
display(L2.bracket(L0).display(M_left_inv_frame))
display(L2.bracket(L1).display(M_left_inv_frame))
```

$$[L_0, L_1] = 0$$

$$[L_2, L_0] = L_0$$

$$[L_2, L_1] = L_1$$

Para hacer las cuentas más sencillas (y breves) consideramos métricas diagonales.

[152]:
```
mu, nu, eta = var("mu, nu, eta")
assume(mu>0,nu>0,eta>0)
g_aux = M.tensor_field(0,2,[[mu,0,0],[0,nu,0],[0,0,eta]],
                       frame=M_left_inv_frame,name='g_a')
g[:] = g_aux[:]
display(g.display())
g.display(M_left_inv_frame)
```

$$g = \mu e^{(-2t)} dx \otimes dx + \nu e^{(-2t)} dy \otimes dy + \eta dt \otimes dt$$

[152]:

$$g = \mu L^0 \otimes L^0 + \nu L^1 \otimes L^1 + \eta L^2 \otimes L^2$$

[153]:
```
%time Riem = g.riemann()
```

```
CPU times: user 347 ms, sys: 305 ms, total: 652 ms
Wall time: 1.2 s
```

Con esto ya podemos calcular la curvatura seccional y verificar que M es un espacio de curvatura constante negativa.

[154]:
```
u0,u1,u2,v0,v1,v2 = var("u0,u1,u2,v0,v1,v2")
u = u0 * L0.at(p) + u1 * L1.at(p) + u2 * L2.at(p)
v = v0 * L0.at(p) + v1 * L1.at(p) + v2 * L2.at(p)
curv_secc(u, v)
```

[154]:

$$-\frac{1}{\eta}$$

19.4. Ejemplo: curvatura positiva. Este caso lo vamos a dejar como un ejercicio para quien esté interesado. La idea es la siguiente. Consideramos la inmersión $\mathbb{S}^3 \hookrightarrow \mathbb{R}^4$. La métrica en $\mathbb{S}^3$ es la inducida de $\mathbb{R}^4$, vía la identificación

$$T_p\mathbb{S}^3 = \{p\}^\perp = \{v \in \mathbb{R}^4 : \langle v, p\rangle = 0\}$$

Con esta métrica la esfera (de radio 1, centrada en el origen) es un espacio de curvatura constate 1. Pero además la esfera es un grupo de Lie: si identificamos $\mathbb{R}^4$ con los cuaterniones $\mathbb{H}$, entonces

$$\mathbb{S}^3 = \{p \in \mathbb{H} : |p| = 1\}$$

es un subgrupo con la multiplicación (y resulta grupo de Lie porque la multiplicación en $\mathbb{H}$ es diferenciable). En $\mathbb{S}^3$ hay un frame invariante a izquierda. En efecto, si $X, Y, Z \in \mathfrak{X}(\mathbb{H})$ son los campos

$$X_p = \mathbf{i}p \qquad Y_p = \mathbf{j}p \qquad Z_p = \mathbf{k}p$$

entonces X, Y, Z son tangentes a la esfera y de hecho su restricción a $\mathbb{S}^3$ forma un frame invariante a izquierda (porque multiplicar por $\mathbf{i}, \mathbf{j}, \mathbf{k}$ son isometrías en $\mathbb{R}^4$).

Ahora sólo falta implementar esto en Sage. Algunas ideas que pueden servir para hacer esto.

[155]:
```
#A partir de la version 9.3, la esfera ya viene precargada
S3 = manifolds.Sphere(3)
print(S3)
```

```
3-sphere S^3 of radius 1 smoothly embedded in the 4-dimensional
  Euclidean space E^4
```

Por defecto $\mathbb{S}^3$ viene con coordenadas esféricas, pero también podríamos pedirle a Sage que use las proyecciones estereográficas

[156]:
```

```

```
%display plain
iota = S3.embedding()
iota.display()
```

[156]:
```
iota: S^3 → E^4
on A: (chi, theta, phi) ↦ (x1, x2, x3, x4) =
  (cos(phi)*sin(chi)*sin(theta), sin(chi)*sin(phi)*sin(theta),
  cos(theta)*sin(chi), cos(chi))
```

Podemos inducir la métrica de $\mathbb{E}^4$ a la esfera:

[157]:
```
%display latex
gamma = S3.induced_metric()
print(gamma)
gamma.display()
```

```
Riemannian metric gamma on the 3-sphere S^3 of radius 1 smoothly
embedded in the 4-dimensional Euclidean space E^4
```

[157]:

$$\gamma = \mathrm{d}\chi \otimes \mathrm{d}\chi + \sin(\chi)^2\,\mathrm{d}\theta \otimes \mathrm{d}\theta + \sin(\chi)^2 \sin(\theta)^2\,\mathrm{d}\phi \otimes \mathrm{d}\phi$$

Los campos X, Y, Z en $\mathbb{E}^4$ son fáciles de definir. Por ejemplo

$$X = -x_2\frac{\partial}{\partial x_1} + x_1\frac{\partial}{\partial x_2} - x_4\frac{\partial}{\partial x_3} + x_3\frac{\partial}{\partial x_4}$$

Solo faltaría definir sus restricciones a la esfera y probar que son invariantes a izquierda.

Agradecimiento

Quisiera agradecer a la organización del XVI Congreso Dr. Antonio Monteiro, y en especial a Emilio Lauret, por la invitación a dictar este curso.

Referencias

[dC92] M. P. do Carmo, *Riemannian Geometry*, Mathematics: Theory & Applications, Birkhäuser, Boston, MA, 1992. MR 1138207.

[Hel78] S. Helgason, *Differential Geometry, Lie Groups, and Symmetric Spaces*, Pure and Applied Mathematics, 80, Academic Press, New York, 1978. MR 0514561.

[KN69] S. Kobayashi and K. Nomizu, *Foundations of Differential Geometry I, II*, Interscience Tracts in Pure and Applied Mathematics, No. 15, Interscience Publishers, New York, 1963, 1969. MR 0152974, MR 0238225.

(Silvio Reggiani) CONICET and Universidad Nacional de Rosario, ECEN–FCEIA, Departamento de Matemática, Av. Pellegrini 250, 2000 Rosario, Argentina

Email address: reggiani@fceia.unr.edu.ar