Riemannian Geometry

RiemannianGeometry is a name space for a series of packages developed in the Department of Geometry and Topology of the University of Santiago de Compostela by Miguel Brozos Vázquez, José Carlos Díaz Ramos, Eduardo García Río and Ramón Vázquez Lorenzo.

These are the packages included:

RiemannianGeometry`Curvature`: a basic package for calculating certain curvature objects in a pseudo-Riemannian manifold.

RiemannianGeometry`DifferentialOperators`: an extension of the basic package that calculates certain differential operators.

RiemannianGeometry`CurvatureOperators`: an extension of the previous package that calculates curvature related objects.

RiemannianGeometry`ExponentialMap`: a package to numerically evaluate the exponential map of a Riemannian manifold and plot geodesics and geodesic spheres.

RiemannianGeometry`CelestialSpheres`: a package to plot geodesic celestial spheres and related objects in a Lorentzian manifold.

RiemannianGeometry`GeodesicSpheres`: a package to calculate power series expansion of geometric objects associated with geodesic spheres.

Apart from these packages, we also include:

RiemannianGeometry`GeometricObjects`: a package containing all the previous packages and some additional functions often used in Riemannian manifolds. This package should be used to introduce new functions by a user trying to extend functionality.

RiemannianGeometry`Notation`: a notational package whose employment is potential unsafe, but that introduces convenient shortcuts in most situations.

The basic Curvature package

A theorem of Whitney states that every differentiable manifold can be embedded in an Euclidean space of sufficiently large dimension. So, we can consider a manifold, at least locally, as the image of a smooth map, $\phi : U \to \mathbb{R}^n$, where U is an open set in certain Euclidean space.

This package provides functionality to compute objects associated with the curvature of a general semi-Riemannian manifold defined as the image of a parametrization in an Euclidean space. It is also the basic package for many other packages that calculate more complicated objects.

`CoordinateVector[`*i*`][`*M*`][`*m*`]`	computes the *i*th coordinate vector field of the manifold *M* at the point *m*
`CoordinateVectors[`*M*`][`*m*`]`	computes the coordinate vector fields of the manifold *M* at the point *m*
`Metric[`*M*`][`*m*`]`	computes the metric of the manifold *M* at *m*. The result is given as a list
`ChristoffelSymbols[`*M*`][`*m*`]`	computes the Christoffel symbols of *M* at *m*. The result is given as a nested list
`ScalarCurvature[`*M*`][`*m*`]`	calculates the scalar curvature of *M* at *m*

Basic geometric objects in a manifold.

All manifolds have to be defined previously as a function or be given as a pure function. In this case it does not matter whether the manifold evaluates its arguments to a number or not.

This loads the package.

```
In[1]:= Needs["RiemannianGeometry`Curvature`"];
```

Here it is the definition of a torus.

```
In[2]:= torus[R_, r_][u_, v_] = {(R + r Cos[u]) Cos[v], (R + r Cos[u]) Sin[v], r Sin[u]};
```

Evaluate the following cell to see the coordinate vector fields, the metric induced on the torus, its Christoffel symbols and its scalar curvature (which is essentially its Gaussian curvature).

```
In[3]:= CoordinateVector[1][torus[R, r]][u, v]
        CoordinateVector[2][torus[R, r]][u, v]
        CoordinateVectors[torus[R, r]][u, v]
        Metric[torus[R, r]][u, v] // Simplify // MatrixForm
        ChristoffelSymbols[torus[R, r]][u, v] // Simplify
        ScalarCurvature[torus[R, r]][u, v] // Simplify
```

```
Out[3]= {-r Cos[v] Sin[u], -r Sin[u] Sin[v], r Cos[u]}
```

```
Out[4]= {-(R + r Cos[u]) Sin[v], (R + r Cos[u]) Cos[v], 0}
```

```
Out[5]= {{-r Cos[v] Sin[u], -r Sin[u] Sin[v], r Cos[u]}, {-(R + r Cos[u]) Sin[v], (R + r Cos[u]) Cos[v], 0}}
```

$$\text{Out[6]//MatrixForm} = \begin{pmatrix} r^2 & 0 \\ 0 & (R + r\,\text{Cos}[u])^2 \end{pmatrix}$$

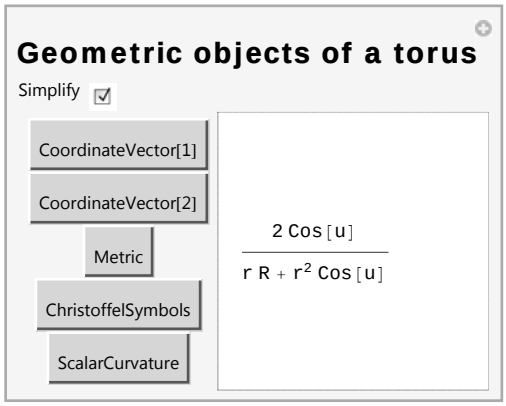
$$\text{Out[7]} = \left\{\left\{\{0, 0\}, \left\{0, -\frac{r\,\text{Sin}[u]}{R + r\,\text{Cos}[u]}\right\}\right\}, \left\{\left\{0, -\frac{r\,\text{Sin}[u]}{R + r\,\text{Cos}[u]}\right\}, \left\{\frac{(R + r\,\text{Cos}[u])\,\text{Sin}[u]}{r}, 0\right\}\right\}\right\}$$

$$\text{Out[8]} = \frac{2\,\text{Cos}[u]}{r\,R + r^2\,\text{Cos}[u]}$$

One can also calculate these geometric objects in a more interactive way using functions such as `Manipulate`:

In[9]:=
```
Manipulate[simpl[f[torus[R, r]][u, v]],
 Style["Geometric objects of a torus", "Section"],
 {{simpl, Simplify, "Simplify"}, Checkbox[##, {Identity, Simplify}] &},
 {{f, Metric, ""}, SetterBar[##, {CoordinateVector[1], CoordinateVector[2],
     Metric, ChristoffelSymbols, ScalarCurvature}, Appearance → "Vertical"] &},
 ControlPlacement → {Top, Top, Left}]
```

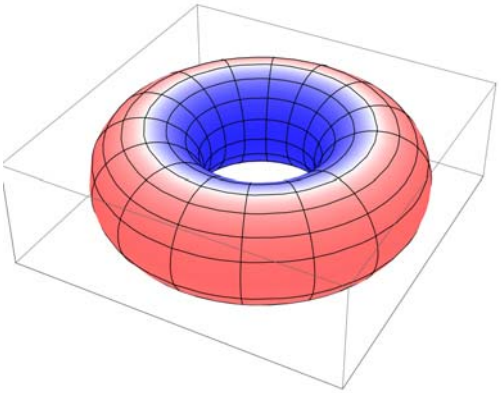
Out[9]=



Evaluate the following lines to plot the torus colored by its scalar (Gaussian) curvature. This time the parameters must be numbers. In the first example we color the torus with a gradient of red when curvature is positive and of blue when it is negative (after rescaling the curvature with `ArcTan` function). In the second case we use the built-in "`TemperatureMap`" gradient to achieve a similar effect.

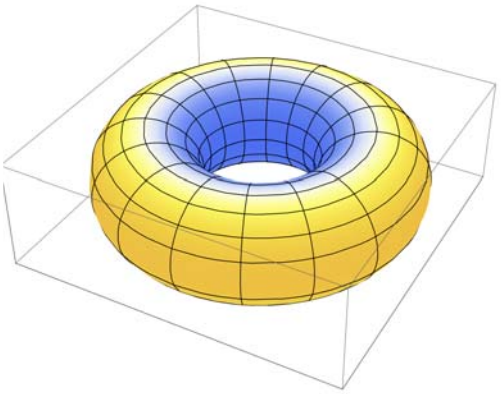
In[10]:=
```
With[{sc = Evaluate[Simplify[ScalarCurvature[torus[2, 1]][#1, #2]]] &},
 ParametricPlot3D[torus[2, 1][u, v], {u, 0, 2 Pi}, {v, 0, 2 Pi},
  ColorFunction → (Glow[Blend[{{-1, RGBColor[0, 0, 1]}, {0, RGBColor[1, 1, 1]}, {1, RGBColor[1, 0, 0]}},
       2.` ArcTan[2.` sc[#4, #5]] / N[Pi]]] &),
  ColorFunctionScaling → False, SphericalRegion → True, Ticks → None]]
```

Out[10]=



```
In[11]:= With[{sc = Evaluate[Simplify[ScalarCurvature[torus[2, 1]][#1, #2]]] &},
 ParametricPlot3D[torus[2, 1][u, v], {u, 0, 2 Pi}, {v, 0, 2 Pi},
  ColorFunction → (Glow[ColorData["TemperatureMap"][0.5 + ArcTan[2.` sc[#4, #5]] / N[Pi]]] &),
  ColorFunctionScaling → False, SphericalRegion → True, Ticks → None]]
```

Out[11]=



Below, we will discuss the definitions and sign conventions that we will use in this package, but first we discuss an important topic.

`Metric[M][m]`	computes the metric of the manifold *M* at *m* when the manifold is given as an embedded submanifold of $\mathbb{R}^n$. The metric can also be provided by user as an up–value of `Metric`
`Metric[M]^=Function[{coords}, {{g11[coords],...,g1n[coords]},..., {g1n[coords],...,gnn[coords]}}]`	uses an up–value to define *a* metric for the manifold *M* when the usual Euclidean metric is not assumed. Then, the user must provide an $n \times n$ matrix *g* depending on the coordinates

Two ways of setting up a metric for a manifold.

If the submanifold is assumed to be (locally) embedded in $\mathbb{R}^n$ by means of a differentiable parametrisation $\phi : U \to \mathbb{R}^n$, where *U* is an open set of $\mathbb{R}^k$, then `Metric` calculates the metric of the manifold *M* assuming that it is the one induced by the usual Euclidean product of $\mathbb{R}^n$. That is, by default the metric tensor is computed as the inner product of the coordinate vector fields.

Nonetheless, one might be interested in considering more general semi-Riemannian manifolds: manifolds embedded in $\mathbb{R}^n$ but with a metric different from that induced by the usual Euclidean inner product. Hence, one might be tempted to override the definition of `Metric`. Note, however, that the symbol `Metric` has the attribute `Protected`, so a down-value cannot be defined directly to it. You should not unprotect this symbol as this could lead to a malfunction of the package. Instead, if you need to define a particular metric for a manifold use an up-value.

Evaluate this cell to define the metric of a pp-wave. We do not need to define the coordinates as they are the usual ones in $\mathbb{R}^4$. The definition is given as an up-value of the symbol `ppW` to avoid associating it with `Metric`.

```
In[12]:= Metric[ppW] ^= Function[{u, v, x, y}, {{-2 H[u, x, y], -1, 0, 0}, {-1, 0, 0, 0}, {0, 0, 1, 0}, {0, 0, 0, 1}}];
```

We can calculate the usual geometric objects:

```
In[13]:= ChristoffelSymbols[ppW][u, v, x, y]
         ScalarCurvature[ppW][u, v, x, y]
```

```
Out[13]= {{{0, H^(1,0,0)[u, x, y], H^(0,1,0)[u, x, y], H^(0,0,1)[u, x, y]}, {0, 0, 0, 0}, {0, H^(0,1,0)[u, x, y], 0, 0}, {0, H^(0,0,1)
          {{0, 0, 0, 0}, {0, 0, 0, 0}, {0, 0, 0, 0}, {0, 0, 0, 0}}, {{0, H^(0,1,0)[u, x, y], 0, 0}, {0, 0, 0, 0}, {0, 0, 0, (
```

```
Out[14]= 0
```

It is important to point out that when defining a metric in this way it is the responsibility of the user to verify that the matrix provided is indeed a semi-Riemannian metric. This means that the matrix must be symmetric and non-degenerate. If these requirements fail to be met then the subsequent results are unpredictable and most likely wrong.

Here is another example: the Poincaré half plane is the set $\{(x, y) \in \mathbb{R}^2 : y > 0\}$ with the Riemannian metric $g(x, y) = \frac{1}{y^2} g_0(x, y)$, being g_0 the usual metric of the 2-dimensional Euclidean space. Here, the definition of the metric is given as a pure function.

```
In[15]:= poincare[u_, v_] = {u, v};
         Metric[poincare] ^= IdentityMatrix[2] / #2^2 &;
```

This gives one of the Christoffel symbols of the Poincaré half plane.

```
In[17]:= ChristoffelSymbols[poincare][u, v][[1, 1, 2]]
```

$$\text{Out[17]} = \frac{1}{v}$$

Of course, the Poincaré half plane has constant negative curvature.

```
In[18]:= ScalarCurvature[poincare][u, v] // Simplify
```

```
Out[18]= -2
```

In a surface, most of the intrinsic geometry is explained by the Gaussian curvature which is up to a constant the scalar curvature. Moreover, the curvature tensor of a surface is determined by the scalar curvature. In manifolds of higher dimension the curvature tensor turns out to be very complicated. As a consequence, other geometric objects are calculated to provide geometric information.

`CurvatureTensor[M][m]`	computes the (1,3) curvature tensor of *M* at *m*, giving the result as a nested list
`CovariantCurvatureTensor[M][m]`	computes the (0,4) curvature tensor of *M* at *m*, giving the result as *a* nested list
`RicciTensor[M][m]`	computes the Ricci tensor of *M* at *m*, giving the result as *a* nested list
`ScalarCurvature[M][m]`	calculates the scalar curvature of *M* at *m*
`WeylTensor[M][m]`	calculates the (0,4) Weyl tensor of *M* at *m*, giving the result as *a* nested list

Basic curvature objects in a manifold.

In what follows we will describe briefly the definitions of the previously defined geometric objects to clarify the sign conventions. Then, we will provide an example of use.

Let *M* be a manifold and suppose that the coordinates are given by $(x^1, \ldots, x^n)$. Let us call *g* its metric tensor. The *Levi-Civita connection*, ∇, is the unique connection on *M* that is torsion-free and makes the metric parallel. It is given by the Koszul formula:

$$g(\nabla_X Y, Z) = \frac{1}{2}(X\, g(Y, Z) + Y\, g(X, Z) - Z\, g(X, Y) - g(X, [Y, Z]) + g(Y, [Z, X]) + g(Z, [X, Y])).$$

where X, Y and Z are vector fields on M and $[\blacksquare, \blacksquare]$ is the Lie bracket, that is, $[X, Y]f = X(Yf) - Y(Xf)$, for a function f.

From now on we denote the coordinate vector fields by $\frac{\partial}{\partial x^1}$, ..., $\frac{\partial}{\partial x^n}$. The *Christoffel symbols*, Γ, are then defined by

$$\nabla_{\frac{\partial}{\partial x^i}} \frac{\partial}{\partial x^j} = \sum_{k=1}^{n} \Gamma_{ij}^{k} \frac{\partial}{\partial x^k}.$$

The *curvature tensor*, R, is the covariant tensor whose coordinates are defined by the following formula

$$R_{ijkl} = g\left(\nabla_{\left[\frac{\partial}{\partial x^i}, \frac{\partial}{\partial x^j}\right]} \frac{\partial}{\partial x^k} - \nabla_{\frac{\partial}{\partial x^i}} \nabla_{\frac{\partial}{\partial x^j}} \frac{\partial}{\partial x^k} + \nabla_{\frac{\partial}{\partial x^j}} \nabla_{\frac{\partial}{\partial x^i}} \frac{\partial}{\partial x^k}, \frac{\partial}{\partial x^l}\right).$$

Then, the *Ricci tensor*, ρ, is the contraction of the curvature tensor in its first and third index and the *scalar curvature*, τ is the contraction of the Ricci tensor. The *Weyl tensor*, W, is defined by the formula in coordinates:

$$W_{ijkl} = R_{ijkl} + \frac{1}{n-2}\left(\rho_{il} g_{jk} + \rho_{jk} g_{ik} - \rho_{jl} g_{ik} - \rho_{ik} g_{jl}\right) - \frac{\tau}{(n-1)(n-2)}\left(g_{il} g_{jk} - g_{ik} g_{jl}\right).$$

This is the curvature tensor of a pp-Wave:

```
In[19]:= CurvatureTensor[ppW][u, v, x, y]
```

```
Out[19]= {{{{0, 0, 0, 0}, {0, 0, 0, 0}, {0, 0, 0, 0}, {0, 0, 0, 0}}, {{0, 0, 0, 0}, {0, 0, 0, 0}, {0, 0, 0, 0}, {0, 0, 0,
    {{0, 0, H^(0,2,0)[u, x, y], H^(0,1,1)[u, x, y]}, {0, 0, 0, 0}, {0, H^(0,2,0)[u, x, y], 0, 0}, {0, H^(0,1,1)[u, x, y], 0,
    {{0, 0, H^(0,1,1)[u, x, y], H^(0,0,2)[u, x, y]}, {0, 0, 0, 0}, {0, H^(0,1,1)[u, x, y], 0, 0}, {0, H^(0,0,2)[u, x, y], 0,
    {{0, 0, 0, 0}, {0, 0, 0, 0}, {0, 0, 0, 0}, {0, 0, 0, 0}}, {{0, 0, 0, 0}, {0, 0, 0, 0}, {0, 0, 0, 0}, {0, 0, 0,
  {{{0, 0, -H^(0,2,0)[u, x, y], -H^(0,1,1)[u, x, y]}, {0, 0, 0, 0}, {0, -H^(0,2,0)[u, x, y], 0, 0}, {0, -H^(0,1,1)[u, x, y]
    {{0, 0, 0, 0}, {0, 0, 0, 0}, {0, 0, 0, 0}, {0, 0, 0, 0}}, {{0, 0, 0, 0}, {0, 0, 0, 0}, {0, 0, 0, 0}, {0, 0, 0,
  {{{0, 0, -H^(0,1,1)[u, x, y], -H^(0,0,2)[u, x, y]}, {0, 0, 0, 0}, {0, -H^(0,1,1)[u, x, y], 0, 0}, {0, -H^(0,0,2)[u, x, y]
    {{0, 0, 0, 0}, {0, 0, 0, 0}, {0, 0, 0, 0}, {0, 0, 0, 0}}, {{0, 0, 0, 0}, {0, 0, 0, 0}, {0, 0, 0, 0}, {0, 0, 0,
```

We can also calculate the (0,4) metric equivalent of this tensor:

```
In[20]:= CovariantCurvatureTensor[ppW][u, v, x, y]
```

```
Out[20]= {{{{0, 0, 0, 0}, {0, 0, 0, 0}, {0, 0, 0, 0}, {0, 0, 0, 0}}, {{0, 0, 0, 0}, {0, 0, 0, 0}, {0, 0, 0, 0}, {0, 0, 0,
    {{0, 0, H^(0,2,0)[u, x, y], H^(0,1,1)[u, x, y]}, {0, 0, 0, 0}, {-H^(0,2,0)[u, x, y], 0, 0, 0}, {-H^(0,1,1)[u, x, y], 0,
    {{0, 0, H^(0,1,1)[u, x, y], H^(0,0,2)[u, x, y]}, {0, 0, 0, 0}, {-H^(0,1,1)[u, x, y], 0, 0, 0}, {-H^(0,0,2)[u, x, y], 0,
  {{{0, 0, 0, 0}, {0, 0, 0, 0}, {0, 0, 0, 0}, {0, 0, 0, 0}}, {{0, 0, 0, 0}, {0, 0, 0, 0}, {0, 0, 0, 0}, {0, 0, 0,
    {{0, 0, 0, 0}, {0, 0, 0, 0}, {0, 0, 0, 0}, {0, 0, 0, 0}}}, {{{0, 0, -H^(0,2,0)[u, x, y], -H^(0,1,1)[u, x, y]}, {0,
    {{0, 0, 0, 0}, {0, 0, 0, 0}, {0, 0, 0, 0}, {0, 0, 0, 0}}, {{0, 0, 0, 0}, {0, 0, 0, 0}, {0, 0, 0, 0}, {0, 0, 0,
  {{{0, 0, -H^(0,1,1)[u, x, y], -H^(0,0,2)[u, x, y]}, {0, 0, 0, 0}, {H^(0,1,1)[u, x, y], 0, 0, 0}, {H^(0,0,2)[u, x, y], 0,
    {{0, 0, 0, 0}, {0, 0, 0, 0}, {0, 0, 0, 0}, {0, 0, 0, 0}}, {{0, 0, 0, 0}, {0, 0, 0, 0}, {0, 0, 0, 0}, {0, 0, 0,
```

This is the Ricci tensor

```
In[21]:= RicciTensor[ppW][u, v, x, y]
```

```
Out[21]= {{H^(0,0,2)[u, x, y] + H^(0,2,0)[u, x, y], 0, 0, 0}, {0, 0, 0, 0}, {0, 0, 0, 0}, {0, 0, 0, 0}}
```

The metric contraction of this tensor is the scalar curvature

```
In[22]:= ScalarCurvature[ppW][u, v, x, y]
```

```
Out[22]= 0
```

Finally, we calculate the (0,4) Weyl curvature tensor of a pp-Wave

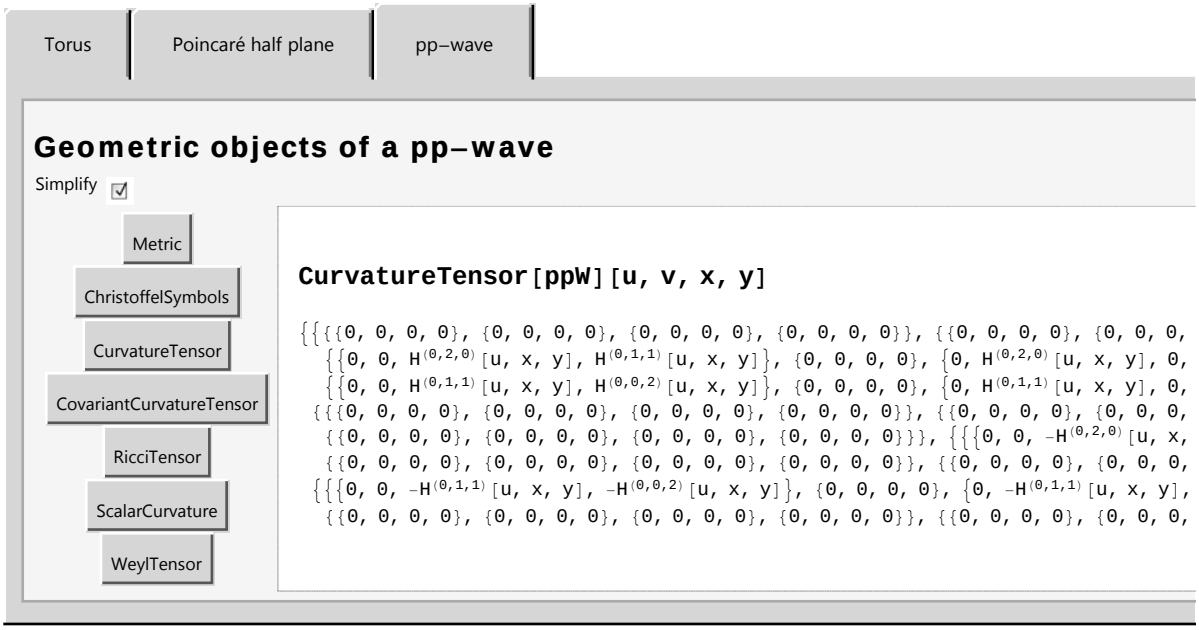
In[23]:=
```
WeylTensor[ppW][u, v, x, y]
```

Out[23]=
$$\left\{\left\{\{\{0,0,0,0\},\{0,0,0,0\},\{0,0,0,0\},\{0,0,0,0\}\},\{\{0,0,0,0\},\{0,0,0,0\},\{0,0,0,0\},\{0,0,0,\right.\right.$$
$$\left\{\left\{0,0,\frac{1}{2}\left(-H^{(0,0,2)}[u,x,y]-H^{(0,2,0)}[u,x,y]\right)+H^{(0,2,0)}[u,x,y],H^{(0,1,1)}[u,x,y]\right\},\{0,0,0,0\},\left\{-H^{(0,2,0)}[u\right.\right.$$
$$\left\{\left\{0,0,H^{(0,1,1)}[u,x,y],H^{(0,0,2)}[u,x,y]+\frac{1}{2}\left(-H^{(0,0,2)}[u,x,y]-H^{(0,2,0)}[u,x,y]\right)\right\},\{0,0,0,0\},\left\{-H^{(0,1,1)}[u\right.\right.$$
$$\{\{\{0,0,0,0\},\{0,0,0,0\},\{0,0,0,0\},\{0,0,0,0\}\},\{\{0,0,0,0\},\{0,0,0,0\},\{0,0,0,0\},\{0,0,0,$$
$$\{\{\{0,0,0,0\},\{0,0,0,0\},\{0,0,0,0\},\{0,0,0,0\}\},\{\{0,0,0,0\},\{0,0,0,0\},\{0,0,0,0\},\{0,0,0,$$
$$\left\{\left\{\left\{0,0,-H^{(0,2,0)}[u,x,y]+\frac{1}{2}\left(H^{(0,0,2)}[u,x,y]+H^{(0,2,0)}[u,x,y]\right),-H^{(0,1,1)}[u,x,y]\right\},\{0,0,0,0\},\left\{\frac{1}{2}\left(-H^{(0,0,\right.\right.\right.$$
$$\{\{0,0,0,0\},\{0,0,0,0\},\{0,0,0,0\},\{0,0,0,0\}\},\{\{0,0,0,0\},\{0,0,0,0\},\{0,0,0,0\},\{0,0,0,$$
$$\left\{\left\{\left\{0,0,-H^{(0,1,1)}[u,x,y],-H^{(0,0,2)}[u,x,y]+\frac{1}{2}\left(H^{(0,0,2)}[u,x,y]+H^{(0,2,0)}[u,x,y]\right)\right\},\{0,0,0,0\},\left\{H^{(0,1,1)}[u,\right.\right.\right.$$
$$\{\{0,0,0,0\},\{0,0,0,0\},\{0,0,0,0\},\{0,0,0,0\}\},\{\{0,0,0,0\},\{0,0,0,0\},\{0,0,0,0\},\{0,0,0,$$

The following command creates a tab viw with the geometric information of the manifolds we have just defined. Note that the Weyl tensor does not make sense for two dimensional manifolds.

In[24]:=
```
TabView[{
  "Torus" → Manipulate[Column[
     {With[{f = f}, Style[HoldForm[f[torus[R, r]][u, v]], Bold, 16]], , simpl[f[torus[R, r]][u, v]]}],
    Style["Geometric objects of the torus", "Section"],
    {{simpl, Simplify, "Simplify"}, Checkbox[##, {Identity, Simplify}] &},
    {{f, Metric, ""}, SetterBar[##, {Metric, ChristoffelSymbols, CurvatureTensor,
        CovariantCurvatureTensor, RicciTensor, ScalarCurvature}, Appearance → "Vertical"] &},
    ControlPlacement → {Top, Top, Left}],
  "Poincaré half plane" → Manipulate[
    Column[{With[{f = f}, Style[HoldForm[f[poincare][u, v]], Bold, 16]], , simpl[f[poincare][u, v]]}],
    Style["Geometric objects of the Poincaré half plane", "Section"],
    {{simpl, Simplify, "Simplify"}, Checkbox[##, {Identity, Simplify}] &},
    {{f, Metric, ""}, SetterBar[##, {Metric, ChristoffelSymbols, CurvatureTensor,
        CovariantCurvatureTensor, RicciTensor, ScalarCurvature}, Appearance → "Vertical"] &},
    ControlPlacement → {Top, Top, Left}],
  "pp-wave" → Manipulate[
    Column[{With[{f = f}, Style[HoldForm[f[ppW][u, v, x, y]], Bold, 16]], , simpl[f[ppW][u, v, x, y]]}],
    Style["Geometric objects of a pp-wave", "Section"],
    {{simpl, Simplify, "Simplify"}, Checkbox[##, {Identity, Simplify}] &},
    {{f, Metric, ""},
     SetterBar[##, {Metric, ChristoffelSymbols, CurvatureTensor, CovariantCurvatureTensor,
        RicciTensor, ScalarCurvature, WeylTensor}, Appearance → "Vertical"] &},
    ControlPlacement → {Top, Top, Left}]}]
```

Out[24]=



Differential operators

The Levi-Civita connection or covariant derivation of a manifold *M* defines a derivation on vector fiels by the well-known Koszul formula:

$$g\left(\nabla_X Y, Z\right) = \frac{1}{2}\left(X\, g(Y,Z) + Y\, g(X,Z) - Z\, g(X,Y) - g(X,[Y,Z]) + g(Y,[Z,X]) + g(Z,[X,Y])\right).$$

This covariant derivative can be extended to other geometric objects and is also used to define more difficult operators. This package loads the basic package `RiemmanianGeometry`Curvature`` and extends its functionallity introducing several differntial operators constructed from the Levi-Civita connection.

`CovariantGradient[`*M*`][`*f*`][`*m*`]`	calculates the gradient of a function *f* in a semi–Riemannian manifold *M* at a point *m*.
`CovariantHessian[`*M*`][`*f*`][`*m*`]`	calculates the Hessian of a function *f* in a semi–Riemannian manifold *M* at a point *m*.
`TensorLaplacian[`*M*`][`*f*`][`*m*`]`	calculates the Laplacian of a function *f* in a semi–Riemannian manifold *M* at a point *m*.

Differential operators on functions.

The *gradient* of a function is the vector field which is dual to the differential of the function, that is, $g(\text{grad}\, f, v) = df(v)$ for any vector *v*, where as usual *g* stands for the metric of the manifold *M*.

The *Hessian* of a function is the second covariant derivative with respect to the Levi-Civita connection, namely, $\nabla^2_{X,Y} f = \nabla_X\left(\nabla_Y f\right)$. Hence, the Hessian of a function is a (0,2) tensor.

The *Laplacian* of a function is the divergence of its gradient, or equivalently, the metric contraction of the Hessian.

If *f* is a function defined on a manifold *M*, then `CovariantGradient`, `CovariantHessian` and `TensorLaplacian` calculate its gradient, Hessian and Laplacian, respectively. The function *f* can be defined previously as a function of the coordinates in *M* or it can be given as a pure function.

First, we load the package

```
In[1]:= Needs["RiemannianGeometry`DifferentialOperators`"];
```

We consider a pp-wave. As usual we usual an up-value to define the metric.

```
In[2]:= Metric[ppW] ^= Function[{u, v, x, y}, {{-2 H[u, x, y], -1, 0, 0},
                                               {-1, 0, 0, 0},
                                               {0, 0, 1, 0},
                                               {0, 0, 0, 1}}];
```

This defines a function:

```
In[3]:= f[u_, v_, x_, y_] = u + v^2;
```

This is its gradient:

```
In[4]:= CovariantGradient[ppW][f][u, v, x, y]
Out[4]= {-2 v, -1 + 4 v H[u, x, y], 0, 0}
```

This is its Hessian

```
In[5]:= CovariantHessian[ppW][f][u, v, x, y]
Out[5]= {{-2 v H^(1,0,0)[u, x, y], 0, -2 v H^(0,1,0)[u, x, y], -2 v H^(0,0,1)[u, x, y]}, {0, 2, 0, 0}, {-2 v H^(0,1,0)[u, x, y], 0, 0,
```

This is the Laplacian

```
In[6]:= TensorLaplacian[ppW][f][u, v, x, y]
Out[6]= 4 H[u, x, y]
```

The covariant derivative can be extended to act not only on functions but also on tensors. Since the metric tensor can be used to rise and lower indexes we will only consider here covariant tensors.

`TensorCovariantDerivative[M][T][m]`	calculates the covariant derivative of the covariant tensor *T* at the point *m* .
`TensorCovariantDerivative[M][T, k][m]`	calculates the *k*th covariant derivative of the covariant tensor *T* at the point *m*.
`TensorLaplacian[M][T][m]`	calculates the covariant Laplacian of the covariant tensor *T* at the point *m*.

Differential operators on covariant tensors.

The extension of the *covariant derivative* to a derivation of a covariant tensor, ω, of order k is defined as follows:

$$\left(\nabla_X \omega\right)_{Y_1,\ldots,Y_k} = X\,\omega_{Y_1\ldots Y_k} - \sum_{i=1}^{k} \omega_{Y_1\ldots \nabla_X Y_i \ldots Y_k},$$

where $X, Y_1, \ldots, Y_k$ are vector fields on M. It can be proved that this gives a covariant tensor of order $k+1$, which is denoted by $\nabla\omega$. As a consequence it can be derived to get covariant derivatives of higher order. As in mathematical notation, in our package, the first index of the corresponding tensor corresponds to the last derivative.

In the case of a general covariant tensor field the *Laplacian* consists of contracting the indexes of the second covariant derivative of that tensor. The Laplacian of a tensor is another tensor of the same order. As it was stated above, if *T* is a function we retrieve its usual Laplacian, which is another function.

Again, note that the tensor *T* can be given as a pure function or it can be defined previously (which is usually more comfortable).

This is the covariant derivative of the Ricci tensor

```
In[7]:= TensorCovariantDerivative[ppW][RicciTensor[ppW]][u, v, x, y]
Out[7]= {{{H^(1,0,2)[u, x, y] + H^(1,2,0)[u, x, y], 0, 0, 0}, {0, 0, 0, 0}, {0, 0, 0, 0}, {0, 0, 0, 0}}, {{0, 0, 0, 0}, {0, 0,
  {{H^(0,1,2)[u, x, y] + H^(0,3,0)[u, x, y], 0, 0, 0}, {0, 0, 0, 0}, {0, 0, 0, 0}, {0, 0, 0, 0}}, {{H^(0,0,3)[u, x, y] + H
```

This is the second covariant derivative of the Ricci tensor. Note here that `TensorCovariantDerivative[ppW][RicciTensor[ppW],2][u,v,x,y][[i,j,k,l]]` corresponds in mathematical notation with $\left(\nabla^2_{ij}\rho\right)_{kl}$.

```
In[8]:= TensorCovariantDerivative[ppW][RicciTensor[ppW], 2][u, v, x, y]
Out[8]= {{{{-H^(0,0,1)[u, x, y] (H^(0,0,3)[u, x, y] + H^(0,2,1)[u, x, y]) - H^(0,1,0)[u, x, y] (H^(0,1,2)[u, x, y] + H^(0,3,0)[u, x, y]) +
    {0, 0, 0, 0}, {0, 0, 0, 0}, {0, 0, 0, 0}, {0, 0, 0, 0}}, {{H^(1,1,2)[u, x, y] + H^(1,3,0)[u, x, y], 0, 0, 0}, {0, 0
   {{H^(1,0,3)[u, x, y] + H^(1,2,1)[u, x, y], 0, 0, 0}, {0, 0, 0, 0}, {0, 0, 0, 0}, {0, 0, 0, 0}}},
  {{{0, 0, 0, 0}, {0, 0, 0, 0}, {0, 0, 0, 0}, {0, 0, 0, 0}}, {{0, 0, 0, 0}, {0, 0, 0, 0}, {0, 0, 0, 0}, {0, 0, 0,
   {{H^(1,1,2)[u, x, y] + H^(1,3,0)[u, x, y], 0, 0, 0}, {0, 0, 0, 0}, {0, 0, 0, 0}, {0, 0, 0, 0}}, {{0, 0, 0, 0}, {0, 0
   {{H^(0,2,2)[u, x, y] + H^(0,4,0)[u, x, y], 0, 0, 0}, {0, 0, 0, 0}, {0, 0, 0, 0}, {0, 0, 0, 0}}, {{H^(0,1,3)[u, x, y] +
  {{{H^(1,0,3)[u, x, y] + H^(1,2,1)[u, x, y], 0, 0, 0}, {0, 0, 0, 0}, {0, 0, 0, 0}, {0, 0, 0, 0}}, {{0, 0, 0, 0}, {0, 0
   {{H^(0,1,3)[u, x, y] + H^(0,3,1)[u, x, y], 0, 0, 0}, {0, 0, 0, 0}, {0, 0, 0, 0}, {0, 0, 0, 0}}, {{H^(0,0,4)[u, x, y] +
```

We can also calculate the Laplacian of the Ricci tensor.

```
In[9]:= TensorLaplacian[ppW][RicciTensor[ppW]][u, v, x, y]
Out[9]= {{H^(0,0,4)[u, x, y] + 2 H^(0,2,2)[u, x, y] + H^(0,4,0)[u, x, y], 0, 0, 0}, {0, 0, 0, 0}, {0, 0, 0, 0}, {0, 0, 0, 0}}
```

Although vector fields can be consider as contravariant tensors and hence, by lowering indexes one can calculate the corresponding metric duals of their differential operators, it is handy to have explicit functions for them as they are very commonly used in Riemannian geometry.

`CovariantDivergence[M][V][m]`	calculates the divergence of *V* at the point *m*.
`VectorCovariantDerivative[M][V][m]`	calculates the total covariant derivative of *V* at the point *m*.
`VectorCovariantDerivative[M][V1, V2][m]`	calculates the covariant derivative of *V2* with respect to *V1*.

Differential operators on vector fields.

The *divergence* of a vector field is the contraction of the total covariant derivative. Namely, $\mathrm{div}\, X = C\, \nabla X$. The result is therefore a function.

The covariant derivative of a vector field is defined by the Koszul formula. The function `VectorCovariantDerivative` can be used in two different ways. The first one, `VectorCovariantDerivative[M][V][m]` calculates the *total covariant derivative* of *V*, which in mathematical notation is usually represented by ∇X. Hence, ∇X is a (1,1)-tensor. As usual, the first entry stands for the derivative (with respect to the coordinate vector fields).

The second form of `VectorCovariantDerivative[M][V1,V2][m]` is the most common one. It calculates the *covariant derivative* of V_2 with respect to V_1, that is, $\nabla_{V_1} V_2$. Hence, the result is another vector field.

As usual, the vector fields can be given as pure functions.

This is the divergence of the gradient of the above function

```
In[10]:= CovariantDivergence[ppW][CovariantGradient[ppW][f]][u, v, x, y]
Out[10]= 4 H[u, x, y]
```

Of course, the divergence of the gradient is the Laplacian.

```
In[11]:= TensorLaplacian[ppW][f][u, v, x, y]
Out[11]= 4 H[u, x, y]
```

The vector field $\frac{\partial}{\partial v}$ is parallel as the following calculation shows. Note that this vector field is given as a pure function.

```
In[12]:= VectorCovariantDerivative[ppW][{0, 1, 0, 0} &][u, v, x, y]
Out[12]= {{0, 0, 0, 0}, {0, 0, 0, 0}, {0, 0, 0, 0}, {0, 0, 0, 0}}
```

We define two vector fields:

```
In[13]:= U[u_, v_, x_, y_] = {1, 0, 0, 0};
         A[u_, v_, x_, y_] = {0, 0, y, -x};
```

This is the covariant derivative of *A* with respect to *U*.

```
In[15]:= VectorCovariantDerivative[ppW][U, A][u, v, x, y]
Out[15]= {0, -x H^(0,0,1)[u, x, y] + y H^(0,1,0)[u, x, y], 0, 0}
```

Although the functions provided in this section use the fact that *M* is (pseudo)-Riemannian manifold, we are now going to introduce three functions that do not use this underlying structure. These are the exterior derivative of forms, the Lie bracket and the Lie derivatives of tensors. These three operators use the differential structure of the manifold (which is essentially the coordinate representation of *M*) but not its metric, so in what follows, the manifold *M* is not passed as an argument.

`ExteriorDerivative[F][m]`	calculates the exterior derivative of the form *F*.
`LieBracket[V1, V2][m]`	calculates the Lie bracket of the tensor fields *V1* and *V2*.
`TensorLieDerivative[V, T][m]`	calculates the Lie derivative of the covariant tensor field *T* with respect to the vector field *V*.

The exterior derivative and the Lie derivatives.

Let F be a p-form, that is, a covariant tensor of order p, which is skew-symmetric. This implies that if σ is a permutation of the set $\{1, \ldots, p\}$, then $F(X_{\sigma(1)}, \ldots, X_{\sigma(p)}) = \text{sgn}(\sigma) F(X_1, \ldots, X_p)$, for all vector fields $X_1, \ldots, X_p$ and where $\text{sgn}(\sigma)$ is the signature of σ. The *exterior derivative* of F is thus defined as

$$dF(X_0, \ldots, X_p) = \sum_{i=0}^{p} (-1)^{i-1} X_i F\left(X_0, \ldots, \hat{X}_i, \ldots, X_p\right) + \sum_{i<j} (-1)^{i+j} F\left(\left[X_i, X_j\right], X_0, \ldots, \hat{X}_i, \ldots, \hat{X}_j, \ldots, X_p\right).$$

As it has already been seen, the *Lie bracket* of two tensor fields X and Y is defined by the formula $[X, Y] f = X(Yf) - Y(Xf)$. Then $[X, Y]$ is again a vector filed on the manifold.

The *Lie derivative*, $\mathcal{L}$, is also a derivation that extends the Lie braket. It can be defined as

$$(\mathcal{L}_X \omega)_{Y_1 \ldots Y_k} = X\, \omega_{Y_1 \ldots Y_k} - \sum_{i=1}^{k} \omega_{Y_1 \ldots [X, Y_i] \ldots Y_k}.$$

for $X, Y_1, \ldots, Y_k$ vector fields on M. Unfortunately, $\mathcal{L}\omega$, is not a tensor as it is not linear in the first variable. That is why in our package the direction of derivation must be explicitly given.

This is the exterior derivative of the function *f* defined above.

```
In[16]:= ExteriorDerivative[f][u, v, x, y]
Out[16]= {1, 2 v, 0, 0}
```

Of course $d^2 = 0$.

```
In[17]:= ExteriorDerivative[ExteriorDerivative[f]][u, v, x, y]
Out[17]= {{0, 0, 0, 0}, {0, 0, 0, 0}, {0, 0, 0, 0}, {0, 0, 0, 0}}
```

Also, the covariant derivative of function coincides with the covarint derivative, although this is no longer true for forms.

```
In[18]:= TensorCovariantDerivative[ppW][f][u, v, x, y]
Out[18]= {1, 2 v, 0, 0}
```

This is the covariant derivative of *A* with respect to *U*.

```
In[19]:= LieBracket[U, A][u, v, x, y]
Out[19]= {0, 0, 0, 0}
```

The following line must be zero because the Levi-Civita connection is torsion-free

```
In[20]:= VectorCovariantDerivative[ppW][A, U][u, v, x, y] -
          VectorCovariantDerivative[ppW][U, A][u, v, x, y] - LieBracket[A, U][u, v, x, y]
Out[20]= {0, 0, 0, 0}
```

The vector field $\frac{\partial}{\partial u}$ is not in general a Killing vector field, because $\mathcal{L}_{\frac{\partial}{\partial u}} g \neq 0$.

```
In[21]:= TensorLieDerivative[U, Metric[ppW]][u, v, x, y]
Out[21]= {{-2 H^(1,0,0)[u, x, y], 0, 0, 0}, {0, 0, 0, 0}, {0, 0, 0, 0}, {0, 0, 0, 0}}
```

Again, the Lie derivative of a function coincides with its differential.

```
In[22]:= TensorLieDerivative[U, f][u, v, x, y]
Out[22]= 1

In[23]:= U[u, v, x, y].ExteriorDerivative[f][u, v, x, y]
Out[23]= 1
```

Curvature operators

The curvature of a pseudo-Riemannian manifold is one the most important objects in psedo-Riemannian geometry. However, it is a very complicated object and is usually the case that other objects, instead of the curvature operator itself are studied. One of the most famous of these objects is the sectional curvature, which essentially contains the same information as the curvature tensor but presented in a different form.

`SectionalCurvature[M][m][x, y]`	calculates the sectional curvature of the manifold *M* at the point *m* with respect to the plane expanded by *x* and *y*.

The sectional curvature of a psedo-Riemannian manifold.

With our sign convention for the curvature tensor, the *sectional curvature K* of a plane π is defined by

$$K(\pi) = \frac{R_{XYXY}}{g(X,X)\,g(Y,Y) - g(X,Y)^2},$$

where $\{X, Y\}$ is a basis of π. Note that in the general psedo-Riemannian setting the plane π must be nondegenerate so that this defintion makes sense. Also take note that this package does not check whether the basis provided by the user is actually a suitable basis (that is, the package does not check whether $\{X, Y\}$ expands a nondegenerate plane).

First, we load the package

```
In[1]:= Needs["RiemannianGeometry`CurvatureOperators`"];
```

This metric defines a Lorentzian manifold of constant sectional curvature *c*.

```
In[2]:= Metric[const[c_]] ^= Function[{t, x, y},
          1/(1 + c/4 (-t^2 + x^2 + y^2))^2 {{-1, 0, 0}, {0, 1, 0}, {0, 0, 1}}];
```

We check that the curvature depends neither on the base point nor on the tangent plane.

```
In[3]:= SectionalCurvature[const[c]][t, x, y][Array[u, 3], Array[v, 3]] // Simplify
Out[3]= c
```

Now, we introduce other curvature operators that are usually studied in pseudo-Riemannian geometry. These operator are somehow employed to study the "eigenvalue structure" of the curvature tensor.

`JacobiOperator[M][m][v]`	calculates the Jacobi operator of *M* at *m* with respect to the direction *v*.
`RicciOperator[M][m]`	calculates the Ricci operator at the point *m*.
`SkewSymmetricCurvatureOperator[M][m][v1, v2]`	calculates the skew–symmetric operator of *M* at the point *m* with respect to the plane spanned by *v1* and *v2*.
`SzaboOperator[M][m][v]`	calculates the Szabo operator of *M* at the point *m* with respect to the direction *v*.

Some curvature operators of a Riemannian manifold.

Given a vector field *X* the *Jacobi operator*, *J*, associated with *X* is the endomorphism defined as

$$J_X(Y) = R_{XY}\,X = \nabla_{[X,Y]}\,X - \nabla_X\,\nabla_Y\,X + \nabla_Y\,\nabla_X\,X.$$

Because of the identities of the curvature tensor one usually restricts it as $J_X : X^\perp \to X^\perp$. However, for computational reasons we do not do this here.

The *Ricci operator* Q is by defintion the operator which is metrically equivalent to the Ricci tensor, that is, $g(Q(X), Y) = \rho(X, Y)$, where ρ is the Ricci tensor.

The *skew-symmetric curvature operator* $R(\pi)$ of an oriented plane π is defined by $g(R(\pi)\,X, Y) = R_{E_1 E_2 X Y}$, where $\{E_1, E_2\}$ is an oriented orthonormal basis of π. As usual, this package does not check whether the vectors supplied by the user actually form an oriented orthonormal basis.

Finally, the *Szabo operator* is defined as the endomorphism S_X such that $S_X(Y) = \left(\nabla_X R\right)_{XY} X$.

This is the Jacobi operator o a manifold of constant sectional curvature with rspect to an arbitrary vector $(\alpha_1, \alpha_2, \alpha_3)$.

```
In[4]:= JacobiOperator[const[c]][t, x, y] @@ Array[α, 3] // Simplify
```

```
Out[4]= {{16 c (α[2]^2 + α[3]^2)/(-4 + c (t^2 - x^2 - y^2))^2, -(16 c α[1] α[2])/(-4 + c (t^2 - x^2 - y^2))^2,
    -(16 c α[1] α[3])/(-4 + c (t^2 - x^2 - y^2))^2},
   {16 c α[1] α[2]/(-4 + c (t^2 - x^2 - y^2))^2, 16 c (-α[1]^2 + α[3]^2)/(-4 + c (t^2 - x^2 - y^2))^2,
    -(16 c α[2] α[3])/(-4 + c (t^2 - x^2 - y^2))^2}, {16 c α[1] α[3]/(-4 + c (t^2 - x^2 - y^2))^2, -(16 c α[2
```

Of course, a manifold of constant sectional curvature is Osserman:

```
In[5]:= Eigenvalues[JacobiOperator[const[c]][t, x, y] @@ Array[α, 3] /
          Array[α, 3].Metric[const[c]][t, x, y].Array[α, 3]] // Simplify
```

```
Out[5]= {0, c, c}
```

The Ricci operator is then a multiple of the identity.

```
In[6]:= RicciOperator[const[c]][t, x, y] // Simplify
```

```
Out[6]= {{2 c, 0, 0}, {0, 2 c, 0}, {0, 0, 2 c}}
```

The vectors $\left(1+\frac{1}{4}c\left(-t^2+x^2+y^2\right)\right)\partial_1$ and $\left(1+\frac{1}{4}c\left(-t^2+x^2+y^2\right)\right)\partial_3$ are orthonormal:

```
In[7]:= {1 + 1/4 c (-t^2 + x^2 + y^2), 0, 0}.Metric[const[c]][t, x, y].{1 + 1/4 c (-t^2 + x^2 + y^2), 0, 0} // Simplify
```

```
Out[7]= -1
```

```
In[8]:= {0, 0, 1 + 1/4 c (-t^2 + x^2 + y^2)}.Metric[const[c]][t, x, y].{0, 0, 1 + 1/4 c (-t^2 + x^2 + y^2)} // Simplify
```

```
Out[8]= 1
```

```
In[9]:= {1 + 1/4 c (-t^2 + x^2 + y^2), 0, 0}.Metric[const[c]][t, x, y].{0, 0, 1 + 1/4 c (-t^2 + x^2 + y^2)} // Simplify
```

```
Out[9]= 0
```

This is the skew-symmetric curvature operator with respect to the previous vectors.

```
In[10]:= SkewSymmetricCurvatureOperator[const[c]][t, x, y][
           {1 + 1/4 c (-t^2 + x^2 + y^2), 0, 0}, {0, 0, 1 + 1/4 c (-t^2 + x^2 + y^2)}] // Simplify
```

```
Out[10]= {{0, 0, -c}, {0, 0, 0}, {-c, 0, 0}}
```

Manifolds of constant sectional curvature are symmetric, so their Szabo operator must vanish

In[11]:=
```
SzaboOperator[const[c]][t, x, y] @@ Array[α, 3] // Simplify
```

Out[11]=
```
{{0, 0, 0}, {0, 0, 0}, {0, 0, 0}}
```

We now consider a pp-wave metric.

In[12]:=
```
Metric[ppW] ^= Function[{u, v, x, y}, {{-2 H[u, x, y], -1, 0, 0}, {-1, 0, 0, 0}, {0, 0, 1, 0}, {0, 0, 0, 1}}];
```

This is the Jacobi operator of a pp-wave metric with respect to an arbitrary vector $(\alpha_1, \alpha_2, \alpha_3, \alpha_4)$.

In[13]:=
```
JacobiOperator[ppW][u, v, x, y] @@ Array[α, 4]
```

Out[13]=
```
{{0, 0, 0, 0}, {α[4] (-α[4] H^(0,0,2)[u, x, y] - α[3] H^(0,1,1)[u, x, y]) + α[3] (-α[4] H^(0,1,1)[u, x, y] - α[3] H^(0,2,0)[u
  0, α[1] α[4] H^(0,1,1)[u, x, y] + α[1] α[3] H^(0,2,0)[u, x, y], α[1] α[4] H^(0,0,2)[u, x, y] + α[1] α[3] H^(0,1,1)[u, x, y]
 {α[1] (-α[4] H^(0,1,1)[u, x, y] - α[3] H^(0,2,0)[u, x, y]), 0, α[1]^2 H^(0,2,0)[u, x, y], α[1]^2 H^(0,1,1)[u, x, y]}, {α[1] (
```

This is the Ricci operator of a pp-wave.

In[14]:=
```
RicciOperator[ppW][u, v, x, y] // Simplify
```

Out[14]=
```
{{0, 0, 0, 0}, {-H^(0,0,2)[u, x, y] - H^(0,2,0)[u, x, y], 0, 0, 0}, {0, 0, 0, 0}, {0, 0, 0, 0}}
```

The vectors ∂_3 and ∂_4 are orthonormal:

In[15]:=
```
{0, 0, 1, 0}.Metric[ppW][u, v, x, y].{0, 0, 1, 0} // Simplify
```

Out[15]=
```
1
```

In[16]:=
```
{0, 0, 0, 1}.Metric[ppW][u, v, x, y].{0, 0, 0, 1} // Simplify
```

Out[16]=
```
1
```

In[17]:=
```
{0, 0, 1, 0}.Metric[ppW][u, v, x, y].{0, 0, 0, 1} // Simplify
```

Out[17]=
```
0
```

This is the skew-symmetric curvature operator with respect to the previous vectors.

In[18]:=
```
SkewSymmetricCurvatureOperator[ppW][u, v, x, y][{0, 0, 1, 0}, {0, 0, 1, 0}] // Simplify
```

Out[18]=
```
{{0, 0, 0, 0}, {0, 0, 0, 0}, {0, 0, 0, 0}, {0, 0, 0, 0}}
```

Manifolds of constant sectional curvature are symmetric, so their Szabo operator must vanish

In[19]:=
```
SzaboOperator[ppW][u, v, x, y] @@ Array[α, 4] // Simplify
```

Out[19]=
```
{{0, 0, 0, 0},
 {-α[4]^3 H^(0,0,3)[u, x, y] - 3 α[3] α[4]^2 H^(0,1,2)[u, x, y] - 3 α[3]^2 α[4] H^(0,2,1)[u, x, y] - α[3]^3 H^(0,3,0)[u, x, y] - α[
  0, α[1] (α[4]^2 H^(0,1,2)[u, x, y] + 2 α[3] α[4] H^(0,2,1)[u, x, y] + α[3]^2 H^(0,3,0)[u, x, y] + α[1] α[4] H^(1,1,1)[u, x, y]
  α[1] (α[4]^2 H^(0,0,3)[u, x, y] + 2 α[3] α[4] H^(0,1,2)[u, x, y] + α[3]^2 H^(0,2,1)[u, x, y] + α[1] α[4] H^(1,0,2)[u, x, y] + α
 {-α[1] (α[4]^2 H^(0,1,2)[u, x, y] + 2 α[3] α[4] H^(0,2,1)[u, x, y] + α[3]^2 H^(0,3,0)[u, x, y] + α[1] α[4] H^(1,1,1)[u, x, y] +
  α[1]^2 (α[4] H^(0,2,1)[u, x, y] + α[3] H^(0,3,0)[u, x, y] + α[1] H^(1,2,0)[u, x, y]), α[1]^2 (α[4] H^(0,1,2)[u, x, y] + α[3] H
 {-α[1] (α[4]^2 H^(0,0,3)[u, x, y] + 2 α[3] α[4] H^(0,1,2)[u, x, y] + α[3]^2 H^(0,2,1)[u, x, y] + α[1] α[4] H^(1,0,2)[u, x, y] +
  α[1]^2 (α[4] H^(0,1,2)[u, x, y] + α[3] H^(0,2,1)[u, x, y] + α[1] H^(1,1,1)[u, x, y]), α[1]^2 (α[4] H^(0,0,3)[u, x, y] + α[3] H
```

Geometric objects in a Riemannian manifold

This package is an extension of the previous packages: `RiemannianGeometry`Curvature``, `RiemannianGeometry`DifferentialOperators`` and `CurvatureOperators`CurvatureOperators``.

The purpose of this package is to include these other packages to allow the user to handle all the functionally provided them, as well as to extend these packages by introducing new functions here. `RiemannianGeometry`GeometricObjects`` is intended to be a starting point for those user trying to write more functions for handling pseudo-Riemannian manifolds. Those new functions must be included here and tested thoroughly before being included in the other packages. This will ensure a clean structure, a trustful behaviour and the possibility to use write other functions for the set of packages `RiemannianGeometry`` without having to pay attention to malfunction due to the use of different versions.

New functions must be added and documented properly using the standards of *Mathematica* packages to `GeometricObjects.nb` in the root directory of this set of packages. The functons must be documented as well in this file, located at `Documentation/English/GeometricObjects.nb`.

Please, include your new functons in what follows.

`SuperRicciTensor[M][m]`	calculates the super Ricci tensor of the manifold *M* at *m*.

The super Ricci tensor.

`GaussianCurvature[S][x, y]`	calculates the Gaussian curvature of the surface *S* at {*x*,*y*}.
`GaussMap[S][x, y]`	calculates the Gaussian map of the surface *S* at {*x*,*y*}.
`MeanCurvature[S][x, y]`	calculates the mean curvature of the surface *S* at {*x*,*y*}.

Some objects on surfaces.

Please, include here an example of use.

First, we load the package

```
In[1]:= Needs["RiemannianGeometry`GeometricObjects`"];
```

Here it is the definition of a torus.

```
In[2]:= torus[R_, r_][u_, v_] = {(R + r Cos[u]) Cos[v], (R + r Cos[u]) Sin[v], r Sin[u]};
```

The super-Ricci tensor of a surface is always a multiple of the metric:

```
In[3]:= SuperRicciTensor[torus[R, r]][u, v] // Simplify
```

$$\text{Out[3]}= \left\{\left\{\frac{2\,\text{Cos}[u]^2}{(R + r\,\text{Cos}[u])^2}, 0\right\}, \left\{0, \frac{2\,\text{Cos}[u]^2}{r^2}\right\}\right\}$$

The Gaussian curvature is essentially the scalar curvature

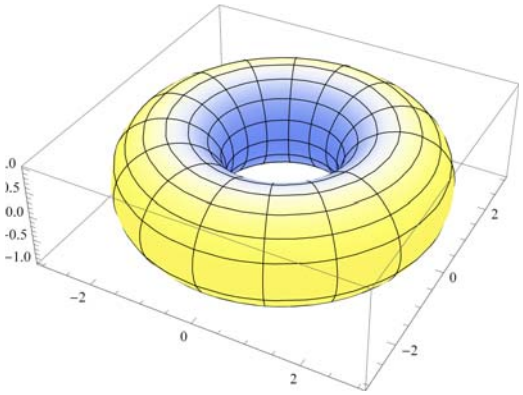
```
In[4]:= GaussianCurvature[torus[R, r]][u, v] // Simplify
```

$$\text{Out[4]}= \frac{\text{Cos}[u]}{r\,R + r^2\,\text{Cos}[u]}$$

This plots a torus and colours it by its Gaussian curvature (using the "TemperatureMap" colour gradient).

In[5]:=
```
With[{sc = Evaluate[Simplify[GaussianCurvature[torus[2, 1]][#1, #2]]] &},
 ParametricPlot3D[torus[2, 1][u, v], {u, 0, 2 Pi}, {v, 0, 2 Pi}, ColorFunction →
   (Glow[ColorData["TemperatureMap"][0.5 + ArcTan[2.` sc[#4, #5]] / Pi]] &), ColorFunctionScaling → False]]
```

Out[5]=



This is the mean curvature of a torus

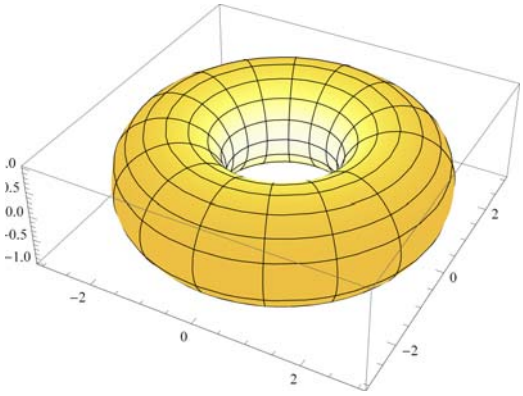
In[6]:=
```
MeanCurvature[torus[R, r]][u, v] // Simplify // PowerExpand
```

Out[6]=
$$\frac{R + 2\,r\,\text{Cos}[u]}{2\,r\,(R + r\,\text{Cos}[u])}$$

This plots a torus and colours it by its mean curvature (using the "TemperatureMap" colour gradient).

In[7]:=
```
With[{sc = Evaluate[Simplify[MeanCurvature[torus[2, 1]][#1, #2]]] &},
 ParametricPlot3D[torus[2, 1][u, v], {u, 0, 2 Pi}, {v, 0, 2 Pi}, ColorFunction →
   (Glow[ColorData["TemperatureMap"][0.5 + ArcTan[2.` sc[#4, #5]] / Pi]] &), ColorFunctionScaling → False]]
```

Out[7]=



This is the Gauss map of a torus

In[8]:=
```
GaussMap[torus[R, r]][u, v] // Simplify // PowerExpand
```

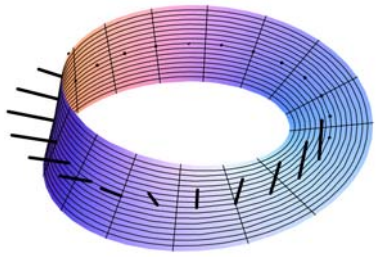
Out[8]=
```
{-Cos[u] Cos[v], -Cos[u] Sin[v], -Sin[u]}
```

This is a graphic proof that the Moebius strip is non-orientable:

In[9]:=
```
moebiusStrip[a_][u_, v_] = a {Cos[u] + v Cos[u / 2] Cos[u], Sin[u] + v Cos[u / 2] Sin[u], v Sin[u / 2]};
```

In[10]:=
```
Show[Graphics3D[{Thick, Line[Table[{N[moebiusStrip[3][u, 0]],
      N[moebiusStrip[3][u, 0] + GaussMap[moebiusStrip[3]][u, 0]]}, {u, 0, 2 π, 2 π/24}]]}],
 ParametricPlot3D[moebiusStrip[3][u, v], {u, 0, 2 π}, {v, -0.3`, 0.3`}, PlotPoints → {25, 7}],
 Boxed → False, Axes → False]
```

Out[10]=



Notation of geometric objects in a Riemannian manifold

This is a package that introduces some notation for the geometric objects described so far. It should be used with care since the symbols introduced here are quite common and thus they are very like to produce clashes with other packages.

Notation	Name of function
Γ	ChristoffelSymbols
Rc	CovariantCurvatureTensor
R	CurvatureTensor
ρ	RicciTensor
τ	ScalarCurvature
K	SectionalCurvature
W	WeylTensor
∇	TensorCovariantDerivative
$\triangle$	TensorLaplacian
grad	CovariantGradient
div	CovariantDivergence
d	ExteriorDerivative
$\pounds$	TensorLieDerivative

List of symbols used for the usual geometric objects in a pseudo-Riemannian manifold.

This packages includes the functionallity provided by RiemannianGeometry`Curvature`, RiemannianGeometry`DifferentialOperators`, RiemannianGeometry`CurvatureOperators` and RiemannianGeometry`GeometricObjects`. Its implementation is very simple: it just associates the symbols on left-hand side of the above table with the functions on the right-hand side. The user of this set of packages may modify this file at convenience to provide shortcuts to any function.

The standard version of this package also contains a palette to use the above notation. This palette is opened automatically when loaded. You can also obtain it by clicking here.

First, we load the package

```
In[1]:= Needs["RiemannianGeometry`Notation`"];
```

We consider a pp-wave:

```
In[2]:= Metric[ppW] ^= Function[{u, v, x, y}, {{-2 H[u, x, y], -1, 0, 0}, {-1, 0, 0, 0}, {0, 0, 1, 0}, {0, 0, 0, 1}}];
```

These are the Christoffel symbols:

```
In[3]:= Γ[ppW][u, v, x, y]
```

```
Out[3]= {{{0, H^(1,0,0)[u, x, y], H^(0,1,0)[u, x, y]}, {H^(0,1,0)[u, x, y], 0, 0}, {H^(0,0,1)[u, x, y], 0, 0}}, {{0, 0, 0, 0}, {0, 0, 0, 0}, {0, 0, 0, 0}, {0, 0, 0, 0}}, {{0, H^(0,1,0)[u, x, y], 0, 0}, {0, 0, 0, 0}, {0, 0, 0, 0}, {0, 0, 0, 0}}, {{0, H^(0,0,1)[u, x, y], 0, 0}, {0, 0, 0, 0}, {0, 0, 0, 0}, {0, 0, 0, 0}}}
```

This is the (0,4) curvature tensor.

```
In[4]:= Rc[ppW][u, v, x, y]
```

```
Out[4]= {{{{0, 0, 0, 0}, {0, 0, 0, 0}, {0, 0, 0, 0}, {0, 0, 0, 0}}, {{0, 0, 0, 0}, {0, 0, 0, 0}, {0, 0, 0, 0}, {0, 0, 0, 0}}, {{0, 0, H^(0,2,0)[u, x, y], H^(0,1,1)[u, x, y]}, {0, 0, 0, 0}, {-H^(0,2,0)[u, x, y], 0, 0, 0}, {-H^(0,1,1)[u, x, y], 0, 0, 0}}}, {{{0, 0, 0, 0}, {0, 0, 0, 0}, {0, 0, 0, 0}, {0, 0, 0, 0}}, {{0, 0, 0, 0}, {0, 0, 0, 0}, {0, 0, 0, 0}, {0, 0, 0, 0}}, {{0, 0, -H^(0,2,0)[u, x, y], -H^(0,1,1)[u, x, y]}, {0, 0, 0, 0}, {H^(0,2,0)[u, x, y], 0, 0, 0}, {H^(0,1,1)[u, x, y], 0, 0, 0}}, {{0, 0, 0, 0}, {0, 0, 0, 0}, {0, 0, 0, 0}, {0, 0, 0, 0}}}, {{{0, 0, -H^(0,1,1)[u, x, y], -H^(0,0,2)[u, x, y]}, {0, 0, 0, 0}, {H^(0,1,1)[u, x, y], 0, 0, 0}, {H^(0,0,2)[u, x, y], 0, 0, 0}}, {{0, 0, 0, 0}, {0, 0, 0, 0}, {0, 0, 0, 0}, {0, 0, 0, 0}}, {{0, 0, 0, 0}, {0, 0, 0, 0}, {0, 0, 0, 0}, {0, 0, 0, 0}}, {{0, 0, 0, 0}, {0, 0, 0, 0}, {0, 0, 0, 0}, {0, 0, 0, 0}}}}
```

This is the (1,3) curvature tensor.

```
In[5]:= R[ppW][u, v, x, y]
```

```
Out[5]= {{{{0, 0, 0, 0}, {0, 0, 0, 0}, {0, 0, 0, 0}, {0, 0, 0, 0}}, {{0, 0, 0, 0}, {0, 0, 0, 0}, {0, 0, 0, 0}, {0, 0, 0,
    {{0, 0, H^(0,2,0)[u, x, y], H^(0,1,1)[u, x, y]}, {0, 0, 0, 0}, {0, H^(0,2,0)[u, x, y], 0, 0}, {0, H^(0,1,1)[u, x, y], 0,
  {{{0, 0, 0, 0}, {0, 0, 0, 0}, {0, 0, 0, 0}, {0, 0, 0, 0}}, {{0, 0, 0, 0}, {0, 0, 0, 0}, {0, 0, 0, 0}, {0, 0, 0,
  {{{0, 0, -H^(0,2,0)[u, x, y], -H^(0,1,1)[u, x, y]}, {0, 0, 0, 0}, {0, -H^(0,2,0)[u, x, y], 0, 0}, {0, -H^(0,1,1)[u, x, y]
    {{0, 0, 0, 0}, {0, 0, 0, 0}, {0, 0, 0, 0}, {0, 0, 0, 0}}}, {{{0, 0, -H^(0,1,1)[u, x, y], -H^(0,0,2)[u, x, y]}, {0,
    {{0, 0, 0, 0}, {0, 0, 0, 0}, {0, 0, 0, 0}, {0, 0, 0, 0}}, {{0, 0, 0, 0}, {0, 0, 0, 0}, {0, 0, 0, 0}, {0, 0, 0,
```

The Ricci tensor.

```
In[6]:= ρ[ppW][u, v, x, y]
```

```
Out[6]= {{H^(0,0,2)[u, x, y] + H^(0,2,0)[u, x, y], 0, 0, 0}, {0, 0, 0, 0}, {0, 0, 0, 0}, {0, 0, 0, 0}}
```

The scalar curvature.

```
In[7]:= τ[ppW][u, v, x, y]
```

```
Out[7]= 0
```

The sectional curvature with respect to an arbitrary plane is rather complicated in thsi case.

```
In[8]:= K[ppW][u, v, x, y][Array[α, 4], Array[β, 4]]
```

```
Out[8]= (α[4] β[1] (α[4] β[1] H^(0,0,2)[u, x, y] + α[3] β[1] H^(0,1,1)[u, x, y] + α[1] (-β[4] H^(0,0,2)[u, x, y] - β[3] H^(0,1,1)[u, x
    α[1] (β[4] (-α[4] β[1] H^(0,0,2)[u, x, y] - α[3] β[1] H^(0,1,1)[u, x, y] + α[1] (β[4] H^(0,0,2)[u, x, y] + β[3] H^(0,1,1)[
  (-((-2 H[u, x, y] α[1] - α[2]) β[1] - α[1] β[2] + α[3] β[3] + α[4] β[4])^2 + (α[1] (-2 H[u, x, y] α[1] - α[2]) - α[1]
```

This is the (0,4) Weyl tensor.

```
In[9]:= W[ppW][u, v, x, y]
```

```
Out[9]= {{{{0, 0, 0, 0}, {0, 0, 0, 0}, {0, 0, 0, 0}, {0, 0, 0, 0}}, {{0, 0, 0, 0}, {0, 0, 0, 0}, {0, 0, 0, 0}, {0, 0, 0,
    {{0, 0, 1/2 (-H^(0,0,2)[u, x, y] - H^(0,2,0)[u, x, y]) + H^(0,2,0)[u, x, y], H^(0,1,1)[u, x, y]}, {0, 0, 0, 0}, {-H^(0,2,0)[u
    {{0, 0, H^(0,1,1)[u, x, y], H^(0,0,2)[u, x, y] + 1/2 (-H^(0,0,2)[u, x, y] - H^(0,2,0)[u, x, y])}, {0, 0, 0, 0}, {-H^(0,1,1)[u
  {{{0, 0, 0, 0}, {0, 0, 0, 0}, {0, 0, 0, 0}, {0, 0, 0, 0}}, {{0, 0, 0, 0}, {0, 0, 0, 0}, {0, 0, 0, 0}, {0, 0, 0,
  {{{0, 0, -H^(0,2,0)[u, x, y] + 1/2 (H^(0,0,2)[u, x, y] + H^(0,2,0)[u, x, y]), -H^(0,1,1)[u, x, y]}, {0, 0, 0, 0}, {1/2 (-H^(0,0,
    {{0, 0, 0, 0}, {0, 0, 0, 0}, {0, 0, 0, 0}, {0, 0, 0, 0}}, {{0, 0, 0, 0}, {0, 0, 0, 0}, {0, 0, 0, 0}, {0, 0, 0,
  {{{0, 0, -H^(0,1,1)[u, x, y], -H^(0,0,2)[u, x, y] + 1/2 (H^(0,0,2)[u, x, y] + H^(0,2,0)[u, x, y])}, {0, 0, 0, 0}, {H^(0,1,1)[u,
    {{0, 0, 0, 0}, {0, 0, 0, 0}, {0, 0, 0, 0}, {0, 0, 0, 0}}, {{0, 0, 0, 0}, {0, 0, 0, 0}, {0, 0, 0, 0}, {0, 0, 0,
```

This the covariant derivative of the Ricci tensor.

```
In[10]:= ▽[ppW][ρ[ppW]][u, v, x, y]
```

```
Out[10]= {{{H^(1,0,2)[u, x, y] + H^(1,2,0)[u, x, y], 0, 0, 0}, {0, 0, 0, 0}, {0, 0, 0, 0}, {0, 0, 0, 0}}, {{0, 0, 0, 0}, {0, 0,
  {{H^(0,1,2)[u, x, y] + H^(0,3,0)[u, x, y], 0, 0, 0}, {0, 0, 0, 0}, {0, 0, 0, 0}, {0, 0, 0, 0}}, {{H^(0,0,3)[u, x, y] + H
```

And this is the covariant Laplacian of the Ricci tensor.

```
In[11]:= △[ppW][ρ[ppW]][u, v, x, y]
```

```
Out[11]= {{H^(0,0,4)[u, x, y] + 2 H^(0,2,2)[u, x, y] + H^(0,4,0)[u, x, y], 0, 0, 0}, {0, 0, 0, 0}, {0, 0, 0, 0}, {0, 0, 0, 0}}
```

We define a function for the following calculations.

```
In[12]:= f[u_, v_, x_, y_] = u + v^2;
```

This is its gradient:

```
In[13]:= grad[ppW][f][u, v, x, y]
Out[13]= {-2 v, -1 + 4 v H[u, x, y], 0, 0}
```

This is the Laplacian

```
In[14]:= △[ppW][f][u, v, x, y]
Out[14]= 4 H[u, x, y]
```

The divergence of the gradient is the Laplacian

```
In[15]:= div[ppW][grad[ppW][f]][u, v, x, y]
Out[15]= 4 H[u, x, y]
```

This is the exterior derivative of the function *f* defined above.

```
In[16]:= d[f][u, v, x, y]
Out[16]= {1, 2 v, 0, 0}
```

Of course $d^2 = 0$.

```
In[17]:= d[d[f]][u, v, x, y]
Out[17]= {{0, 0, 0, 0}, {0, 0, 0, 0}, {0, 0, 0, 0}, {0, 0, 0, 0}}
```

This is the Lie derivative of the metric with respect to the vector field ∂_1:

```
In[18]:= ℒ[{1, 0, 0, 0} &, Metric[ppW]][u, v, x, y]
Out[18]= {{-2 H^(1,0,0)[u, x, y], 0, 0, 0}, {0, 0, 0, 0}, {0, 0, 0, 0}, {0, 0, 0, 0}}
```

Explorador de superficies.nbp *

superficie Superficie aleatoria

alturas

nodos

Representación gráfica | Animacións | Xeodésicas | Dúas xeodésicas

Superficie aleatoria

{x, y}→{x, y, InterpolatingFunction[{{−2.1, 2.1}, {−2.1, 2.1}}, <>][x, y]}

Transparencia

Xeodésicas

Punto

Dirección

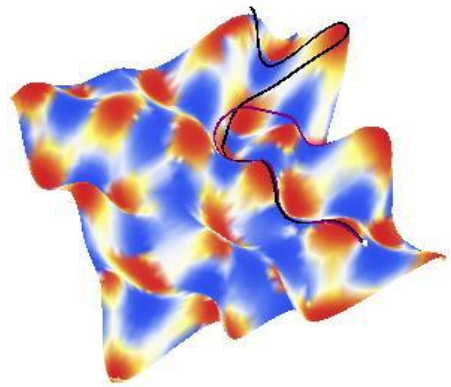
Dirección

Separación

Lonxitude

Lonxitude real: 14.1014

(A lonxitude non varía linearmente)



100%