Bimetric computations in xAct 2

Manifold and chart

Manifold $\mathcal{M}$ and chart $\mathcal{B}$ should be defined!

Define metrics

```
Quiet@Remove["E▮", "L▮", "S▮", "A▮", "η▮"]
```

Minkowski metric

```
η▮ = DiagonalMatrix[ {-1} ~Join~ ConstantArray[ 1, DimOfManifold[M]-1 ] ];
```

Metric g⌘ (printed as *g*)

We first define the global ambient metric. An induced metric is then obtained by projection of ambient metric along hypersurface-orthogonal vector fields.

```
Print[ xAct`xCore`Private`bars ]
```

```
If[ NameQ["g⌘"],
    Quiet@DefMetric[ -1, g⌘[-a, -b], CD, PrintAs → "g", SymbolOfCovD → {";", "∇g" } ]
]
```

Define *g* through a vielbein:

```
If[ Length[E▮] =!= 0,
    g▮ = E▮ᵀ.η▮.E▮ //Simplify;
]
```

We use the function `MetricInBasis` to input the components of the metric in a given basis. We simply pass the components matrix as third argument. Note the minus sign in front of $\mathcal{B}$. This indicates that we are supplying the covariant components of the metric.

```
MetricInBasis[ g⌘, -ℬ, g■ ];
```

```
MetricCompute[ g⌘, ℬ, "Metric"[-1, -1], CVSimplify → Simplify ]
MetricCompute[ g⌘, ℬ, "Metric"[1, 1], CVSimplify → Simplify ]
MetricCompute[ g⌘, ℬ, "DetMetric"[], CVSimplify → Simplify ]
```

Metric f⌘ (printed as *f*)

A metric is always defined on a given vbundle (that of its abstract indices at definition time), which is stored as an upvalue for the function `VBundleOfMetric`. However, a vbundle can have several metrics (stored in the function `MetricsOfVBundle`). A vbundle with at least one metric gives `True` under the function `MetricEndowedQ`, and `False` if it has not got any metric.

If there are several metrics **only the first one will be used to raise and lower indices**; all other metrics are called "frozen" and do not have all the expected properties for the first-metric. In particular, the inverse of a frozen metric `frozen[-a, -b]` is not `frozen[a, b]` (which is actually `g[a, c] g[b, d] frozen[-c, -d]`, with *g* being the first-metric), but is defined as `Invfrozen[a, b]`, using the head `Inv`.

```
Print[ xAct`xCore`Private`bars ]
```

```
Quiet@DefMetric[ -1, f⌘[-a,-b], CDf, PrintAs → "f", SymbolOfCovD → { "|", "∇f" } ]
```

Define *f* through a vielbein:

```
If[ Length[L■] =!= 0,
    f■ = L■ᵀ.η■.L■ //Simplify;
]
```

```
MetricInBasis[ f⌘, -ℬ, f■ ];
```

The square root $\sqrt{g^{-1}\ f}$

Transformation $A = g^{-1} f$

```
Print[ xAct`xCore`Private`bars ]
```

```
Quiet@DefTensor[ A[a,-b], ℳ ]
```

```
A■ = Inverse@g■.f■ //FullSimplify;
```

```
AllComponentValues[ A[{a,ℬ},{-b,-ℬ}], A■ ];
```

The square root $S = \sqrt{A} = \sqrt{g^{-1} f}$

```
Quiet@DefTensor[ S[a,-b], ℳ ]
```

```
If[ Length[S■] === 0,
    S■ = MatrixPower[ A■, 1/2 ] // FullSimplify
];
```

```
If[ Length[L■] === 0 ∧ Length[E■] =!= 0,
    L■ = E■.S■ // Simplify;
];
```

```
AllComponentValues[ S[{a,ℬ},{-b,-ℬ}], S■ ];
```

The inverse of the square root

```
Quiet@DefTensor[ iS[a,-b], ℳ, PrintAs → "⟦S⁻¹⟧" ]
```

```
AllComponentValues[ iS[{a,ℬ},{-b,-ℬ}], Inverse@S▮ ];
```

Metric $h = gS$

```
h▮ = g▮.S▮ // Simplify;
```

Consistency check for elementary symmetric polynomials

```
If[ DimOfManifold[ℳ] === 4, Block[
    {e1,e2,e3,e4},

    e1 = ℰ[S[a,-b],0] == 1;
    e2 = ℰ[S[a,-b],1] == S[a,-a] //ToCanonical;
    e3 = ℰ[S[a,-b],2] == 1/2 (S[a,-a]S[b,-b]-S[a,-b]S[b,-a]) //ToCanonical;
    e4 = ℰ[S[a,-b],3] == 1/6 (S[a,-a]S[b,-b]S[c,-c]-3S[a,-a]S[b,-c]S[c,-b]+2S[a,-b]S[b,-c]S[c,-a]) //ToC

    If[ ¬( e1 ∧ e2 ∧ e3 ∧ e4 ), Print@Style["E for Elementary symmetric polynomials failed",Red]

    e1 = 𝒴[S[a,-b],0] == g⌘[a,-b]//ToCanonical;
    e2 = 𝒴[S[a,-b],1] == S[a,-b]-S[c,-c]g⌘[a,-b]//ToCanonical;
    e3 = 𝒴[S[a,-b],2] == S[a,-c]S[c,-b]-S[a,-b]S[c,-c]+1/2 g⌘[a,-b](S[c,-c]S[d,-d]-S[c,-d]S[d,-c])//ToCa
    e4 = 𝒴[S[a,-b],3] == S[a,-c]S[c,-d]S[d,-b]-S[a,-c]S[c,-b]S[d,-d]+1/2 S[a,-b](S[c,-c]S[d,-d]-S[c,-d]S

If[ ¬( e1 ∧ e2 ∧ e3 ∧ e4 ), Print@Style["𝒴 for Elementary symmetric polynomials failed",Red] ];
]]
```

Compute Riemann, Ricci, Einstein...

Compute all the tensors.

```
?MetricCompute
```

MetricCompute[g, ch, T] computes the components of the curvature tensor T associated to the metric g in the chart ch, where g and ch are symbols already known to xTensor and xCoba, respectively. The metric g is assummed to have been assigned values as explicit functions of the coordinate scalars. The notation for T is special and currently allows the 15 possibilities: "Metric"[–1, –1], "Metric"[1, 1], "DetMetric"[], "DMetric"[–1, –1, –1], "DDMetric"[–1, –1, –1, –1], "Christoffel"[–1, –1, –1], "Christoffel"[1, –1, –1], "Riemann"[–1, –1, –1, –1], "Riemann"[–1, –1, –1, 1], "Riemann"[–1, –1, 1, 1], "Ricci"[–1, –1], "RicciScalar"[], "Weyl"[–1, –1, –1, –1], "Einstein"[–1, –1], "Kretschmann"[], where –1 denotes a covariant component and 1 a contravariant component. This function computes in advance everything needed to know the required tensor T. It is possible to say All instead of a tensor T, and then those 14 tensors will be computed. There are options CVSimplify, to specify a function which is applied to each component after each tensor is computed (default is Together), and Verbose, to get info messages during the computation (default is True).

```
Print[ xAct`xCore`Private`bars ]
```

```
(
    Print[ "** MetricCompute g: ", #1 ];
    MetricCompute[ g⌘, ℬ, #1, CVSimplify → Simplify, Parallelize → True ]
) & /@
{
    "Metric"[-1, -1], "Metric"[1, 1], "DetMetric"[],
    "Christoffel"[-1, -1, -1], "Christoffel"[1, -1, -1],
    "Riemann"[-1, -1, -1, -1], "Riemann"[-1, -1, -1, 1], "Riemann"[-1, -1, 1, 1],
    "Ricci"[-1, -1], "RicciScalar"[], "Einstein"[-1, -1],
    "DMetric"[-1, -1, -1], "DDMetric"[-1, -1, -1, -1],
    "Weyl"[-1, -1, -1, -1], "Kretschmann"[], "CDRiemann"[-1, -1, -1, -1, -1]
};
Print[ xAct`xCore`Private`bars ]
(
    Print[ "** MetricCompute f: ", #1 ];
    MetricCompute[ f⌘, ℬ, #1, CVSimplify → Simplify, Parallelize → True ]
) & /@
{
    "Metric"[-1, -1], "Metric"[1, 1], "DetMetric"[],
    "Christoffel"[-1, -1, -1], "Christoffel"[1, -1, -1],
    "Riemann"[-1, -1, -1, -1], "Riemann"[-1, -1, -1, 1], "Riemann"[-1, -1, 1, 1],
    "Ricci"[-1, -1], "RicciScalar"[], "Einstein"[-1, -1],
    "DMetric"[-1, -1, -1], "DDMetric"[-1, -1, -1, -1],
    "Weyl"[-1, -1, -1, -1], "Kretschmann"[], "CDRiemann"[-1, -1, -1, -1, -1]
};
```

Equations of Motion

Equations of Motion, 3+1 decomposition

Equations

$$G^{(g)}{}_{ab} + \frac{m^d}{m_g{}^{d-2}} V^{(g)}{}_{ab} = \frac{1}{m_g{}^{d-2}} T^{(g)}{}_{ab}$$

$$V^{(g)}{}_{ab} := \sum_{n=0}^{d-1} (-1)^n \beta_n g_{ac} [Y^n(S)]^c{}_b$$

$$[Y^n(S)]^a{}_b = \sum_{k=0}^{n} (-1)^k \mathcal{E}_n(S) \left[S^{n-k}\right]^c{}_b$$

$$[Y^n(S)]^a{}_b = \frac{2}{\sqrt{-g}} g^{ac} \frac{\delta}{\delta g^{cb}} \left(\sqrt{-g}\ \mathcal{E}_n(S)\right)$$

where $S = \sqrt{g^{-1} f}$

After decomposition, we get the 3+1 Einstein system consiting of:

- The evolution equations

$$\partial_t \gamma_{ij} = -2 N K_{ij} + \mathcal{L}_\beta \gamma_{ij} = -2 N K_{ij} + D_i \beta_j + D_j \beta_i$$

$$\partial_t K_{ij} = -D_i D_j N + N\left(R_{ij} + K K_{ij} - 2 K_{ik} K^k{}_j\right) + \mathcal{L}_\beta K_{ij} + \kappa\left(\tfrac{1}{2} \gamma_{ij}(\mathcal{S} - \rho) - \mathcal{S}_{ij}\right)$$

- The constraint equations

$$R + K^2 - K_{ij} K^{ij} = 2 \kappa \rho \quad \text{(Hamiltonian constraint)}$$

$$D_i\left(K^{ij} - \gamma^{ij} K\right) = \kappa p^j \qquad \text{(momentum constraint)}$$

where

$$\rho := n^a n^b T_{ab} \qquad \text{(energy density; aka } E\text{)}$$

$p^i := -\gamma^{ic} n^d T_{cd}$ (momentum density; aka j^i or S^i)

$S_{ij} := \gamma_i{}^c \gamma_j{}^d T_{cd}$ (stress tensor)

Here we have $S := \gamma^{ij} S_{ij}$, and $S - \rho = T = g^{ab} T_{ab}$.

Potential with respect to *g*

```
Print[ xAct`xCore`Private`bars ]
```

```
Quiet@DefTensor[ Vg[-a,-b], M, Symmetric[{-a,-b}], PrintAs → "⟦V(g)⟧" ]
```

```
AllComponentValues[ Vg[{-a,-B},{-b,-B}],
    Module[ { c, d = DimOfManifold[M] },
        Sum[(-1)^n β[n] g⌘[-a,-c] Y[S[c,-b],n], {n,0,d-1}]
    ] // ToBasis[B] // ToBasis[B] // TraceBasisDummy // ComponentArray // ToValues // Simplify
];
```

Potential with respect to *f*

```
Quiet@DefTensor[ Vf[-a,-b], M, Symmetric[{-a,-b}], PrintAs → "⟦V(f)⟧" ]
```

```
AllComponentValues[ Vf[{-a,-B},{-b,-B}],
    Module[{ c, d = DimOfManifold[M] },
        Sum[(-1)^n β[d-n] f⌘[-a,-c] Y[iS[c,-b],n], {n,0,d-1}]
    ] // ToBasis[B] // ToBasis[B] // TraceBasisDummy // ComponentArray // ToValues // Simplify
];
```

```
Print[ xAct`xCore`Private`bars ]
```

Utility to dump all the calculated values...

```
nbDumpAssumptions[ usedFields_ ] := (
    writeCell[ "Metri g and f", "Subsection" ];
    writeCell[ "Fields used in EoM" ];
    usedFields // printNice // myPrint;
    writeCell[ "Parameter ranges"];
    $Assumptions /. And → List // Column // myPrint;
)
```

```
nbDumpMetric[ name_, g⌘_, g∎_ ]:= (
    writeCell[ "Metric " <> name ];
    g⌘[-a,-b] // printMatrixComponents[B];
    g⌘[a,b] // printMatrixComponents[B];
    { name <> " = ",({d#&/@{ScalarsB}}.g∎.Transpose@{d#&/@{ScalarsB}})⟦1,1⟧} //printNice//Row//my
)
```

```
nbDumpInteractionTerm[] := (
    writeCell[ "Square Root", "Subsection" ];
    A[a,-b] // printMatrixComponents[B];
    S[a,-b] // printMatrixComponents[B];
    iS[a,-b] // printMatrixComponents[B];
    g⌘[-a,-c] S[c,-b] // printComponents[B,MatrixForm];
)
```

```
nbDumpGeometryOf[ name_, g⌘_, ChristoffelCDPD𝓑_, RicciScalarCD_, KretschmannCD_, EinsteinCD_ ] :=
    writeCell[ "The geometry of " <> name, "Subsection" ];
    writeCell[ "Christoffel Symbols for " <> name ];
    ChristoffelCDPD𝓑[a, -b, -c] // printNonZeroComponents[𝓑];
    writeCell["Ricci Scalar for " <> name ];
    RicciScalarCD[] // printComponents[𝓑,Expand];
    writeCell["Kretschmann for " <> name ];
    KretschmannCD[] // printComponents[𝓑,Expand];
    writeCell["Einsten Tensor for " <> name ];
    EinsteinCD[-a,-b] // printNonZeroComponents[𝓑,Expand];
    g⌘[a,c] EinsteinCD[-c,-b] // printNonZeroComponents[𝓑,Simplify];
)

nbDumpGeometryOf[ name_, g⌘_, EinsteinCD_ ] := (
    writeCell[ "The geometry of " <> name, "Subsection" ];
    EinsteinCD[-a,-b] // printNonZeroComponents[𝓑,Expand];
    g⌘[a,c] EinsteinCD[-c,-b] // printNonZeroComponents[𝓑,Simplify];
)

nbDumpPotentialFor[ name_, g⌘_, Vg_ ] := (
    writeCell[ "The potential for " <> name, "Subsection" ];
        g⌘[a,c]Vg[-c,-b]// printNonZeroComponents[𝓑,Simplify];
)
```

Utilities for TEX reporting

```
texDumpAssumptions[ usedFields_ ] := log /@ {
    "% --------------------------------------------------",
    "\\section*{Configuration}",
    "\\begin{description}",
    "\\item [{Chart:}] \\ \\{" <>
        StringRiffle[ texNice[#,"$","$"]& /@ {Scalars𝓑}, ", " ] <>
        "\\}",
    "% --------------------------------------------------",
    "\\item [{Fields:}] \\ " <>
    StringRiffle[ texNice[#,"$",""] <>
        (StringRiffle[ texNice[#,"(",")$"]& /@ {Φield}, "," ]) & /@
        usedFields, ", " ],
    "% --------------------------------------------------",
    "\\item [{Parameter range:}] \\ ",
    StringRiffle[ texNice[#,"$","$"]& /@ ($Assumptions /. And → List), ",\\ \\ " ],
    "\\end{description}",
    ""
}
```

```
texDumpMetric[ name_, g⌘_, g∎_ ]:= log /@ {
    "% --------------------------------------------------",
    "\\subsubsection*{Metric $" <> name <> "$:}",
    "\\begin{align}",
    name <> "_{\\mu\\nu} &= ",
    g⌘[-a,-b]// texMatrixInBasis[ℬ,Simplify],
    "\\!,\\quad " <> name <> "^{\\mu\\nu} = ",
    g⌘[a,b]// texMatrixInBasis[ℬ,Simplify],
    "\\!,\\end{align}",
    "% --------------------------------------------------",
    "\\begin{align}",
    name <> " &= ",
    ({ⅆ#&/@{Scalarsℬ}}.g∎.Transpose@{ⅆ#&/@{Scalarsℬ}})⟦1,1⟧//texNice,
    "\\end{align}"
}
texDumpMetric[ name_, g∎_ ]:= log /@ {
    "% --------------------------------------------------",
    "\\subsubsection*{Metric $" <> name <> "$:}",
    "\\begin{align}",
    name <> "_{\\mu\\nu} &= ",
    g∎ // redefineFields // Simplify // texNiceMatrix,
    "\\!,\\quad " <> name <> "^{\\mu\\nu} = ",
    Inverse@g∎ // redefineFields // Simplify // texNiceMatrix,
    "\\!,\\end{align}",
    "% --------------------------------------------------",
    "\\begin{align}",
    name <> " &= ",
    ({ⅆ#&/@{Scalarsℬ}}.g∎.Transpose@{ⅆ#&/@{Scalarsℬ}})⟦1,1⟧ // texNice,
    "\\end{align}"
}
texDumpInteractionTerm[] := log /@ {
    "% --------------------------------------------------",
    "\\section*{Interaction Term}",
    "\\subsubsection*{Square Root $S$, where $S^2 = A := g^{-1}f$}",
    "\\begin{align}",
    "S^{\\mu}{}_{\\nu} &= ",
    S[a,-b] // texMatrixInBasis[ℬ,Simplify],
    "\!,\\quad (S^{-1})^{\\nu}{}_{\\mu} = ",
    iS[a,-b] // texMatrixInBasis[ℬ,Simplify],
    "\\!,\\end{align}",
    "% --------------------------------------------------",
    "\\begin{align}",
    "A^{\\mu}{}_{\\nu} &= ",
    A[a,-b] // texMatrixInBasis[ℬ,Simplify],
    "\\!,\\quad h_{\\mu\\nu} = g_{\\mu\\rho} S^{\\rho}{}_{\\nu} = ",
    g⌘[-a,-c] S[c,-b] // texMatrixInBasis[ℬ,Simplify],
    "\\end{align}"
}
```

```
texDumpGeometryOf[ name_, g⌘_, ChristoffelCDPDℬ_, RicciScalarCD_, KretschmannCD_, EinsteinCD_ ] :
    "% ---------------------------------------------------",
    "\\section*{The Geometry of $" <> name <> "$}",
    "\\subsubsection*{Christoffel Symbols}" ,
    "\\begin{align}",
    ChristoffelCDPDℬ[a, -b, -c] // texNonZeroComponents[ℬ,Expand],
    "\\end{align}",
    "% ---------------------------------------------------",
    "\\subsubsection*{Ricci Scalar}",
    "\\begin{align} R[\\nabla_" <> name <> "] &= ",
    RicciScalarCD[] // texInBasis[ℬ,Expand] // myTexBreak,
    "\\end{align}",
    "% ---------------------------------------------------",
    "\\subsubsection*{Kretschmann Scalar}",
    "\\begin{align} K[\\nabla_" <> name <> "] &= ",
    KretschmannCD[] // texInBasis[ℬ,Expand] // myTexBreak,
    "\\end{align}",
    "% ---------------------------------------------------",
    "\\subsubsection*{Einsten Tensor}",
    "\\begin{align}",
    EinsteinCD[-a,-b] // texNonZeroComponents[ℬ,Expand],
    "\\end{align}",
    "\\begin{align}",
    g⌘[a,c] EinsteinCD[-c,-b] // texNonZeroComponents[ℬ,Expand],
    "\\end{align}"
}
```