Bimetric computations in xAct 1

```
(*********************************************************************)
(* Bimetric computations in xAct. Part I.                            *)
(* Copyright (c) 2014-2015 by Mikica B. Kocic, under GPL.            *)
(*********************************************************************)

(*
  GNU General Public License (GPL)

  This program is free software; you can redistribute it and/or modify
  it under the terms of the GNU General Public License as published
  by the Free Software Foundation; either version 2 of the License,
  or (at your option) any later version.

  This program is distributed in the hope that it will be useful,
  but WITHOUT ANY WARRANTY; without even the implied warranty of
  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU
  General Public License for more details.

  You should have received a copy of the GNU General Public License
  along with this program; if not, write to the Free Software
  Foundation, Inc., 59 Temple Place-Suite 330, Boston, MA 02111-1307,
  USA.
*)
```

Version 1.01 -- 2016-03-04 added Abs[] -> || in print nice

Setup the xAct system

```
Print[ StringRepeat[ "-", 60 ] ]
Print[ Style[ "Loading xAct adapted to bimetric theory...", Blue ] ]
Print[ Style[ "Copyright (C) 2014-2015 by Mikica B. Kocic, under GPL.", Blue ] ]

xAct`Bim`prog$ind = 0; (* reset progress indicator *)
xAct`Bim`time$0 = SessionTime []; (* reset session time *)

If [ ¬NumberQ[xAct`Bim`mem$0],
    xAct`Bim`mem$0  = MemoryInUse [];
    xAct`Bim`cpu$0  = TimeUsed [];
    xAct`Bim`printerConf = 1;
    << xAct`ShowTime1`;
    Utilities`ShowTime`$ShowTimeThreshold = 5;
];
```

Load xAct...

```
BeginPackage[ "xAct`Bim`", { "xAct`xCoba`", "xAct`TexAct`" }]
```

Memory watchdog; forbid *Mathematica* of eating too much memory. Also: Launch all configured parallel subkernels.

```
memoryWatchdog[ mb_ : 1.5 ] := Block[ {},
    Quiet@RemoveScheduledTask[ memory$watchdog ];
    memory$watchdog = RunScheduledTask[
        If[ MemoryInUse[] > mb 1024^3 (* Max 1.5 GB *), Quit[] ],
        1 (* every 1 sec *)
    ];
    Quiet@LaunchKernels [];
]

xAct`xTensor`$Info = False; (* Deactivate info messages in xTensor *)

xAct`xCoba`$CVVerbose = False; (* Deactivate info messages in xCoba *)

$PrePrint = xAct`xTensor`ScreenDollarIndices; (* Hide away ugly dummies *)
```

Display the context and the used resources

```
mem$1 = MemoryInUse [];

cpu$1 = TimeUsed [];

Print[ "Context path: ", Style[ $ContextPath, Blue ] ]
Print[ "Memory in use: ", Style[ mem$1 - mem$0, Blue ] ]
Print[ "Time used: ", Style[ cpu$1 - cpu$0, Blue ] ]
Print[ xAct`xCore`Private`bars ]
```

Nicely print the fields

These rely on the user's function `printNice` that should print scalar fields and their derivatives nicely (devoid of coordinate fields).

```
redefineFields[ expr_ ] := expr;   (* called each time after ToValues[] *)

myPrint = writeExpr;  (* default 'printer' *)

sc$pn[ p_, q_:0, r_:0, s_:0 ] := Style[ ","
    <> If[ p ≤ 0 ⋁ Length[{$ield}] ≤ 0, "", StringRepeat[ ToString[Head[{$ield}⟦1⟧]], p ] ]
    <> If[ q ≤ 0 ⋁ Length[{$ield}] ≤ 0, "", StringRepeat[ ToString[Head[{$ield}⟦2⟧]], q ] ]
    <> If[ r ≤ 0 ⋁ Length[{$ield}] ≤ 0, "", StringRepeat[ ToString[Head[{$ield}⟦3⟧]], r ] ]
    <> If[ s ≤ 0 ⋁ Length[{$ield}] ≤ 0, "", StringRepeat[ ToString[Head[{$ield}⟦4⟧]], s ] ],
    Red
]

printNice[ expr_ ] := expr /. {
    Derivative[2][a_?$ieldQ][Sequence[{Scalars𝒷}⟦1⟧]] :> Style["ä",Red],
    Derivative[2][a_?$ieldQ][Sequence[{Scalars𝒷}⟦2⟧]] :> Style[a'',Red],
    Derivative[1][a_?$ieldQ][Sequence[{Scalars𝒷}⟦1⟧]] :> Style["ȧ",Blue],
    Derivative[1][a_?$ieldQ][Sequence[{Scalars𝒷}⟦2⟧]] :> Style[a',Blue],
    Derivative[p_,q_,r_,s_][a_?$ieldQ][$ield] :> a_sc$pn[p,q,r,s],
    Derivative[p_,q_,r_][a_?$ieldQ][$ield] :> a_sc$pn[p,q,r],
    Derivative[p_,q_][a_?$ieldQ][$ield] :> a_sc$pn[p,q],
    a_?$ieldQ[any__] :> a
}

printComponents[ B_, func_ ][ e_ ] := myPrint[
Row@{
    e, " ≐ ",
    e // ToBasis[B] // ToBasis[B] // TraceBasisDummy // ComponentArray // ToValues // ToValues //
      // printNice // Simplify // func
}]

printComponents[ B_ ][ e_ ] := printComponents[ B, #& ][ e ]
```

```
printMatrixComponents[ B_ ][ e_ ] := myPrint[
Row@{
    e // ToBasis[B], " = ",
    e // ToBasis[B] // ToBasis[B] // xAct`xTensor`ScreenDollarIndices // ComponentArray // ToValu
      // printNice // Simplify // Expand // MatrixForm
}]

printNonZeroComponents[ B_, func_ ][ e_ ] := ( myPrint[#]& /@ (
    Row /@ (
        (
            {
               #1, " ≐ ",
               #1 // TraceBasisDummy // ComponentArray // ToValues // ToValues // redefineFields
                  // printNice // Simplify // func
            }& /@ ( e // ToBasis[B] // ToBasis[B] // xAct`xTensor`ScreenDollarIndices // Componen
        ) // Select[ #1, ( #1[[3]] =!= 0 )& ]&
    )
);)

printNonZeroComponents[ B_ ][ e_ ] := printNonZeroComponents[ B, #& ][ e ]
```

Nicely print the fields in TEX

```
Tex[ Power[ sDot[x_], n_ ] ] := "{\\color{blue}\\dot{" <> Tex@x <> "}^{" <> Tex@n <> "}}"
Tex[ Power[ dDot[x_], n_ ] ] := "{\\color{red}\\ddot{" <> Tex@x <> "}^{" <> Tex@n <> "}}"
Tex[ Power[ sPrime[x_], n_ ] ] := "{\\color{blue}" <> Tex@x <> "^{\\prime\\," <> Tex@n <> "}}"
Tex[ Power[ dPrime[x_], n_ ] ] := "{\\color{red}" <> Tex@x <> "^{\\prime\\prime\\," <> Tex@n <> "

Tex[ sDot[x_]   ] := "{\\color{blue}\\dot{"  <> Tex@x <> "}}"
Tex[ dDot[x_]   ] := "{\\color{red}\\ddot{" <> Tex@x <> "}}"
Tex[ sPrime[x_] ] := "{\\color{blue}" <> Tex@x <> "^{\\prime}}"
Tex[ dPrime[x_] ] := "{\\color{red}" <> Tex@x <> "^{\\prime\\prime}}"

Tex[ subScr[x_,y_?StringQ] ] := Tex@x <> "_{" <> y <> "}"

Tex[ Power[ ⅆx_, n_ ] ] := "\\mathrm{d}" <> Tex@x <> "^{" <> Tex@n <> "}"
Tex[ ⅆx_ ] := "\\mathrm{d}" <> Tex@x

Tex[ x_ ∈ Reals ] := Tex@x <> "\\in\\mathbb{R}"
Tex[ x_ ≠ y_ ] := Tex@x <> "\\ne" <> Tex@y

sc$pnTex[ p_, q_:0, r_:0, s_:0 ] := StringJoin[
    "{\\color{red},",
    If[ p ≤ 0 ⋁ Length[{$ield}] ≤ 0, "", StringRepeat[ " " <> Tex[Head[{$ield}[[1]]], p ] ],
    If[ q ≤ 0 ⋁ Length[{$ield}] ≤ 1, "", StringRepeat[ " " <> Tex[Head[{$ield}[[2]]], q ] ],
    If[ r ≤ 0 ⋁ Length[{$ield}] ≤ 2, "", StringRepeat[ " " <> Tex[Head[{$ield}[[3]]], r ] ],
    If[ s ≤ 0 ⋁ Length[{$ield}] ≤ 3, "", StringRepeat[ " " <> Tex[Head[{$ield}[[4]]], s ] ],
    "}"
]
```

```
tex$Nice[ expr_ ] := expr /. {
    Derivative[2][a_?ΦieldQ][Sequence[{Φield}[[1]]]] :> dDot[a],
    Derivative[2][a_?ΦieldQ][Sequence[{Φield}[[2]]]] :> dPrime[a],
    Derivative[1][a_?ΦieldQ][Sequence[{Φield}[[1]]]] :> sDot[a],
    Derivative[1][a_?ΦieldQ][Sequence[{Φield}[[2]]]] :> sPrime[a],
    Derivative[p_,q_,r_,s_][a_?ΦieldQ][Φield] :> subScr[a,scΦ$pnTex[p,q,r,s]],
    Derivative[p_,q_,r_][a_?ΦieldQ][Φield] :> subScr[a,scΦ$pnTex[p,q,r]],
    Derivative[p_,q_][a_?ΦieldQ][Φield] :> subScr[a,scΦ$pnTex[p,q]],
    a_?ΦieldQ[any__] :> a
}

texNice[ expr_, pre_:"", post_:"" ] := Block[ {s},
    s = expr // tex$Nice // TexPrint;
    pre <> (s // xAct`TexAct`Private`TexFix // StringReplace[#,", "→","]&) <> post
]

texNiceAligned[ expr_, pre_:"", post_:"" ] := Block[ {s},
    s = expr // tex$Nice // TexPrintAlignedEquations;
    pre <> (s // xAct`TexAct`Private`TexFix // StringReplace[#,", "→","]&) <> post
]

texNiceMatrix[ expr_, pre_:"", post_:"" ] := Block[ {s},
    s = expr // tex$Nice // TexMatrix;
    pre <> (s // xAct`TexAct`Private`TexFix // StringReplace[#,", "→","]&) <> post
]

log[ expr_ ] := Block[ {},
    If[ Length[tex$log]===0, tex$log = {} ];
    AppendTo[ tex$log, ToString@expr ];
    expr
]

logNL := (log@"";)

flushTexLog[ filename_:"default-log.tex", tex_:{} ] := Block[
    { str },
    If[ Length[tex] =!= 0,
        str = OpenWrite[ filename ];
        WriteLine[str,#]& /@ {
            "\\documentclass[11pt]{article}", "",
            "\\usepackage[latin9]{inputenc}",
            "\\usepackage[a4paper]{geometry}",
            "\\geometry{verbose,tmargin=2cm,bmargin=2cm,lmargin=2cm,rmargin=2cm}", "",
            "\\usepackage{amsmath,amssymb,bm,color,cancel}", "",
            DateString[{"%%%% Timestamp: ","Year","-","Month","-","Day"," ","Hour",":","Minute","
            "", "\\begin{document}", ""
        };
        WriteLine[str,#]& /@ tex;
        WriteLine[ str, "" ];
        WriteLine[ str, "\\end{document}" ];
        Close[str];
    ]
]
```

```
saveToTexFile[ file_ ][ tex_ ] := Block[ {},
    SetDirectory[ NotebookDirectory[] ];
    flushTexLog[ file <> ".tex", tex ];
    texOut=ReadList[ "!" <> $LatexExecutable <> " " <> file <> ".tex", String ];
    DeleteFile@{ file <> ".log", file <> ".aux" };
    ResetDirectory [];
]

texMatrixInBasis[ B_, func_:(#&) ][ e_ ]:=  texNiceMatrix[
    e // ToBasis[B] // ToBasis[B] // TraceBasisDummy // xAct`xTensor`ScreenDollarIndices
    // ComponentArray // ToValues // redefineFields // Simplify // func
]

texInBasis[ B_, func_:(#&) ][ e_ ]:=  texNice[
    e // ToBasis[B] // ToBasis[B] // TraceBasisDummy // xAct`xTensor`ScreenDollarIndices
    // ComponentArray // ToValues // ToValues // redefineFields // Simplify // func
]

texNonZeroComponents[ B_, func_:(#&) ][ e_ ] := (
  ( StringReplace[ texNice[#[[1]]], "\\nabla " → "\\nabla_" ] <> " &= " <> myTexBreak@texNice[#[[2]]]
    ( {
        #1,
        #1 // TraceBasisDummy // ComponentArray // ToValues // ToValues // redefineFields // Simp
      }& /@ ( e // ToBasis[B] // ToBasis[B] // xAct`xTensor`ScreenDollarIndices // ComponentArray
     ) // Select[ #1, ( #1[[2]] =!= 0 )& ]&
)) // Riffle[ #, "\\\\" ] & ) /. List → Sequence

$TexPrintPageWidth = 400;

If[ ¬MemberQ[ $TexInitLatexPackages, "{color}" ],
    AppendTo[ $TexInitLatexPackages, "{color}" ]
]

myTexBreak[ s_ ] := TexBreak[ s, 400, {},
    TexBreakBy → "TexPoint", TexBreakAt → "+"|"-",
    TexBreakString → "\\nonumber\\\\\n & \\quad"
]
```

General utility functions

```
assumeReal[ v_, cond_: True ] := Block[
    {s = $Assumptions /. And → List },
    If[ ¬MemberQ[ s, v ∈ Reals ], $Assumptions = $Assumptions ⋀ v ∈ Reals ];
    If[ ¬MemberQ[ s, cond ], $Assumptions = $Assumptions ⋀ cond ];
]

UnderBar[ f_ ] := f[ Φield ]

defineField[ f_ ] := (
    ΦieldQ[f] ^= True;
    xAct`xTensor`DefScalarFunction[f]
)

defineField[ f_, options__ ] := (
    ΦieldQ[f] ^= True;
    xAct`xTensor`DefScalarFunction[ f, options ]
)
```

Matrix utility functions

Elementary symmetric polynomials ($0 \le n \le \dim \mathcal{M}$)

$$\mathcal{E}_n(S) := S^{a_1}{}_{[a_1} S^{a_2}{}_{a_2} \ldots S^{a_n}{}_{a_n]}$$

```
ℰ[ S_[a_,-b_], 0 ] := 1

ℰ[ S_[a_,-b_], n_ ] := Block[
    {
        indices = xAct`xTensor`GetIndicesOfVBundle[ TangentM, n, xAct`xTensor`IndicesOfVBundle[Ta
    },
    xAct`xTensor`Antisymmetrize[ Product[ S[z,-z], {z,indices} ], -indices ]
]
```

Matrix power (for integer $n \ge 0$)

$$[\mathcal{P}_n(S)]^a{}_b := S^a{}_{a_1} S^{a_1}{}_{a_2} \ldots S^{a_{n-1}}{}_b$$

```
𝒫[ S_[a_,-b_], 0 ] := 1

𝒫[ S_[a_,-b_], n_ ] := Block[
    {
        iMid = xAct`xTensor`GetIndicesOfVBundle[ TangentM, n-1, Join[ {a,b}, xAct`xTensor`Indices
        iTop,iBot
    },
    iTop = Join[ {a}, iMid ];
    iBot = -Join[ iMid, {b} ];
    Product[
        S[ z /. List → Sequence ],
        {z, Transpose[{iTop,iBot}] }
    ]
]
```

Derivatives of symmetric polynomials of $S = \sqrt{g^{-1} f}$ with respect to g^{ab}

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

```
𝒴[ S_[a_,-b_], n_ ] :=
    Sum[(-1)^k ℰ𝒫[ S[a,-b], n, k ], {k, 0, n}]
```

```
δP[ S_[a_,-b_], n_, k_ ] := Block[
    {
        indices, iMid, iTop, iBot, e, p
    },
    indices = If[ k ≤ 0, {},
        xAct`xTensor`GetIndicesOfVBundle[ TangentM, k, xAct`xTensor`IndicesOfVBundle[TangentM][[1
    ];
    iMid = If[ n ≤ k, {},
        xAct`xTensor`GetIndicesOfVBundle[ TangentM, n-k-1, Join[ {a,b}, indices, xAct`xTensor`Inc
    ];
    iTop = Join[ {a}, iMid ];
    iBot = -Join[ iMid, {b} ];
    e = If[ k ≤ 0, 1,
        xAct`xTensor`Antisymmetrize[ Product[ S[z,-z], {z,indices} ], -indices ]
    ];
    p = If[ n ≤ k, g⌘[a,-b],
        Product[
            S[ z /. List → Sequence ],
            {z, Transpose[{iTop,iBot}] }
        ]
    ];
    e p
]
```

Utilities to save reports

```
createReport[ title_: Null ] := If[ xAct`Bim`printerConf === 2, Block[ { fn },
    If[ (* if does not exist object *)
        ¬ ValueQ@nb ⋁ ( NotebookInformation[nb] // Head ) == Symbol,
    (* Then *)
        nb = CreateDocument [];
        SetOptions[ nb, Magnification → 1.25 ];
        If[ title =!= Null, NotebookWrite[ nb, Cell[ title, "Title" ] ]];
        fn = DateString[{"Created: ","Year","-","Month","-","Day"," ","Hour",":","Minute",":","Se
        NotebookWrite[ nb, Cell[ fn, "Text" ] ],
    (* Else *)
        Null,
    (* Error *)
        SetOptions[ nb, Visible → True ]
    ]
]]

writeExpr[ expr_ ] := Echo[expr]

writeExpr[ expr_ ] := Block[{},
    createReport[];
    NotebookWrite[ nb, Cell[ToBoxes[ExpressionCell[ expr, "Output" ]], "Text" ] ];
] /; xAct`Bim`printerConf === 2

writeExpr[ expr_ ] := expr /; xAct`Bim`printerConf === 3

writeCell[ contents_, style_:"Subsubsection" ] := If[ xAct`Bim`printerConf === 2,
    createReport[];
    NotebookWrite[ nb, Cell[ contents, style ] ];
]
```

End Package

```
EndPackage []
```