Questa versione non va modificata perché rappresenta il punto di partenza degli studi del 2013. Questa versione venne fatta nel giugno 2012.

Libreria mia (v3 - 20110119)

La novità, rispetto alla versione precedente è che uso i TrueTensors ossia i tensori accompagnati dalla tipologia dei loro indici. Definisco "nudi" i tensori privi della specifica del tipo dei loro indici.
Questa libreria è un documento autonomo ossia non fa riferimento a nessun altro documento.
Ho separato le funzioni qui definite, da tutti i test che servono a controllare il loro buon funzionamento ed inoltre ho pensato di realizzare funzioni specifiche per definire particolari metriche importanti.

Questo file va eseguito per primo e poi, in un notebook separato, si possono utilizzare le funzioni qua definite.

```
mialibreria = {}
```

```
{}
```

Libreria di base

- **Trasforma un tensore nudo (privo di specifica del tipo di indici) usando un nuovo sistema di coordinate o lo converte da covariante a controvariante e viceversa**

```
(* tra_ == matrice di trasformazione o tensore metrico nudo *)
(* se è una matrice di trasformazione deve essere la matrice
 che serve per trasformare un vettore controvariante in un nuovo
 vettore controvariante nel nuovo sistema di coordinate *)
(* ten_ == tensore nudo di rango maggiore di 0 da trasformare *)
(* cambia_ == lista di pilotaggio del cambiamento. Se
  un elemento vale 1 allora si applica la trasformazione tra_ ,
se vale -1 allora applica la trasposta dell'inversa di tra_ perché,
nel caso di trasformazione di coordinate, l'indice è covariante,
se vale 0 non fa cambiamenti e questa opzione serve nella
 trasformazione del tipo da covariante a controvariante o viceversa *)
tralba[tra_, ten_, cambia_] := Block[{inTtra, i, ii, n, nc, per, nuovo},
   If[ArrayDepth[tra] ≠ 2, Return["Errore: primo arg non matrice"]];
   If[ArrayDepth[cambia] ≠ 1,
    Return["Errore: secondo argomento non lista"]];
   inTtra = Transpose[Inverse[tra]];
   n = ArrayDepth[ten];
   per = Range[n];
   If[cambia[[1]] > 0,
    nuovo = tra.ten;,
    If[0 > cambia[[1]], nuovo = inTtra.ten;,
     nuovo = ten]];
   If[n == 1, Return[nuovo] ];
   nc = Length[cambia];
   For[i = 2, n ≥ i, i++,
    per[[1]] = i; per[[i]] = 1;
    ii = Min[i, nc];
    If[cambia[[ii]] > 0,
     nuovo = Transpose[tra.Transpose[nuovo, per], per];,
     If[0 > cambia[[ii]],
      nuovo = Transpose[inTtra.Transpose[nuovo, per], per]]];
    per[[1]] = 1; per[[i]] = i];
   Simplify[nuovo]];
mialibreria = Append[mialibreria, "tralba"];
tralba::usage =
  "Trasforma un tensore nudo: cambia sistema di rif. o tipo componenti";
```

- **Derivata ordinaria di un tensore nudo (privo di info sul tipo di indici)**

```
(* ten_ == tensore da derivare *)
(* var_ == lista delle variabili usate *)
derord[ten_, var_] := Block[{i, n, cambia},
   n = ArrayDepth[ten] + 1;
   cambia = Table[Mod[n + i - 2, n] + 1, {i, 1, n}];
   Transpose[Table[D[ten, var[[i]]], {i, 1, Length[var]}], cambia]
  ];
mialibreria = Append[mialibreria, "derord"];
derord::usage = "Derivata ordinaria di un tensore nudo";
```

- **Correzione per ottenere la derivata covariante di un tensore nudo**

```
(* ten_ == tensore nudo di rango maggiore di 0,
di cui calcolo come correggere il suo derivato ordinario *)
(* gamma_ ==
 simboli di Christoffel di seconda specie già calcolati con apposita funzione *)
(* tipo_ == lista del tipo degli indici del tensore ten_ : pari sono covarianti,
dispari sono controvarianti *)
dercov[ten_, gamma_, tipo_] := Block[{ga, nuovo, per, prp, i, n, ii, nc},
   (* aggiunta per derivata covariante *)
   n = ArrayDepth[ten];
   nc = Length[tipo];
   ga = Transpose[gamma];
   ii = Min[n, nc];
   (* pari == covariante, dispari == controvariante *)
   If[Mod[tipo[[ii]], 2] == 0,
    nuovo = -ten.gamma,
    nuovo = ten.ga];
   If[n == 1, Return[nuovo] ];
   per = Range[n];
   prp = Range[n + 1];
   For[i = n - 1, i > 0, i--,
    per[[n]] = i;
    per[[i]] = n;
    prp[[n]] = i;
    prp[[i]] = n;
    ii = Min[i, nc];
    If[Mod[tipo[[ii]], 2] == 0,
     nuovo = nuovo - Transpose[Transpose[ten, per].gamma, prp],
     nuovo = nuovo + Transpose[Transpose[ten, per].ga, prp]
    ];
    per = Range[n];
    prp = Range[n + 1]];
   nuovo];
mialibreria = Append[mialibreria, "dercov"];
dercov::usage = "Calcola la correzione necessaria
    per ottenere la derivata covariante di un tensore nudo";
```

- **Calcola i simboli di Christoffel di seconda specie**

```
(* g222_ == tensore nudo metrico covariante *)
(* lv_ == lista delle variabili *)
faCh3122[g222_, lv_] := Block[{i, g211, g3022, ch3222},
   If[2 > Length[lv], Return["Lista variabili inferiore a 2"]];
   g211 = Simplify[Inverse[g222]];
   g3022 = Table[D[g222, lv[[i]]], {i, 1, Length[lv]}];
   ch3222 = (Transpose[g3022, {3, 1, 2}] + Transpose[g3022, {2, 3, 1}] - g3022) / 2;
   Simplify[Dot[g211, ch3222]]];
mialibreria = Append[mialibreria, "faCh3122"];
faCh3122::usage = "Calcola i simboli di Christoffel di seconda specie";
```

- **Tensore di Riemann col primo indice controvariante**

```
(* ch3122_ == simbolo di Cristoffel di seconda specie *)
(* lv_ == lista variabili *)
faRiemann41222[ch3122_, lv_] := Block[
   {i, ch40122, r1, r2, cx, r3, r4, risulta},
   If[2 > Length[lv], Return["Lista variabili inferiore a 2"]];
   ch40122 = Simplify [Table[D[ch3122, lv[[i]]], {i, 1, Length[lv]}]];
   r1 = Transpose[ch40122, {3, 1, 2, 4}];
   r2 = Transpose[ch40122, {4, 1, 2, 3}];
   cx = Simplify[ch3122.ch3122];
   r3 = Transpose[cx, {1, 3, 2, 4}];
   r4 = Transpose[cx, {1, 4, 2, 3}];
   risulta = r1 - r2 + r3 - r4];
mialibreria = Append[mialibreria, "faRiemann41222"];
faRiemann41222::usage = "Tensore di Riemann di tipo 1222";
```

- **Tensore di Riemann totalmente covariante**

```
(* g222_ == tensore metrico nudo, totalmente covariante *)
(* lv_ == lista delle variabili indipendenti *)
faRiemann42222[g222_, lv_, info_: 0] := Block[
   {i, g3022, g40022, ch3222, tt, ta, tb, ga, gb, gc, gd, risulta},
   If[2 > Length[lv], Return["Lista variabili inferiore a 2"]];
   g3022 = Table[D[g222, lv[[i]]], {i, 1, Length[lv]}];
   If[info ≠ 0, Print["faRiemann42222: fatto g3022"]];
   g40022 = Table[D[g3022, lv[[i]]], {i, 1, Length[lv]}];
   If[info ≠ 0, Print["faRiemann42222: fatto g40022"]];
   ch3222 = (Transpose[g3022, {3, 1, 2}] + Transpose[g3022, {2, 3, 1}] - g3022) / 2;
   If[info ≠ 0, Print["faRiemann42222: fatto ch3222"]];
   tt = Transpose[Inverse[g222].ch3222, {3, 2, 1}].ch3222;
   ta = Transpose[tt, {2, 3, 1, 4}];
   tb = Transpose[tt, {2, 4, 1, 3}];
   ga = Transpose[g40022, {2, 3, 1, 4}];
   gb = Transpose[g40022, {1, 4, 2, 3}];
   gc = Transpose[g40022, {2, 4, 1, 3}];
   gd = Transpose[g40022, {1, 3, 2, 4}];
   risulta (ga + gb - gc - gd) / 2 + ta - tb ];
mialibreria = Append[mialibreria, "faRiemann42222"];
faRiemann42222::usage = "Tensore di Riemann di tipo 2222";
```

- **Tensore di Ricci totalmente covariante**

```
(* g222_ == tensore metrico nudo, totalmente covariante *)
(* lv_ == lista delle variabili indipendenti *)
faRicci222[g222_, lv_, info_: 0] := Block[{ch3122, r41222},
   ch3122 = faCh3122[g222, lv];
   If[info ≠ 0, Print["faRicci222: fatto ch3122"]];
   r41222 = faRiemann41222[ch3122, lv, info];
   If[info ≠ 0, Print["faRicci222: fatto r41222"]];
   Simplify[Tr[Transpose[r41222, {1, 3, 2, 4}], Plus, 2]]];
mialibreria = Append[mialibreria, "faRicci222"];
faRicci222::usage = "Tensore di Ricci di tipo 22";
```

- **Tensore energia impulso dato il tensore elettromagnetico**

Richiede il tensore elettromagnetico in forma totalmente covariante e il tensore metrico in forma totalmente controvariante.

```
(* f222_ == tensore elettromagnetico nudo,
in forma totalmente covariante. Deve essere una matrice antisimmetrica *)
(* gns211_ == tensore metrico nudo in forma totalmente
  controvariante. Deve essere una matrice simmetrica *)
faxt222[f222_, gns211_] := Block[{f222as, g211, t222, tra},
   (* simmetrizza il secondo argomento per sicurezza *)
   f222as = (f222 - Transpose[f222]) / 2;
   g211 = (gns211 + Transpose[gns211]) / 2;
   t222 = Dot[Dot[f222as, g211], Transpose[f222as]];
   tra = Tr[Dot[g211, t222]] / 4;
   t222 = tra Inverse[g211] - t222];
mialibreria = Append[mialibreria, "faxt222"];
faxt222::usage = "Calcola il tensore energia
    impulso nudo assegnato il tensore elettromagnetico nudo";
```

- **Calcolo del tempo locale**

Questa funzione serve a trovare il vettore tempo ossia la combinazione di coordinate che rappresenta la migliore variabile temporale di una data metrica **in un assegnato punto spazio-temporale** ossia in corrispondenza di un dato evento:

Attenzione: tempolocale[g22,tini,itera] **è una funzione numerica** basata sul metodo delle potenze per il calcolo degli autovalori (fa uso del teorema di Gerschgorin) e dunque i valori della matrice g22 (ossia il tensore metrico covariante calcolato in corrispondenza dell'evento calcolato) debbono essere numerici e non simbolici.
Il risultato è la matrice unitaria simmetrica utile per isolare il vero tempo locale dalle altre coordinate spaziali e il vettore unitario stesso...

g22 == una qualsiasi matrice NUMERICA di ordine 4, simmetrica e con determinante negativo.

tini == un vettore che si ritiene una buona approssimazione del vettore cercato. In mancanza di altre informazioni usare il tempo dell' osservatore ossia il vettore {1,0,0,0}.

itera == numero di iterazioni per approssimare il tempo locale. Notare che non occorre in genere trovare il vero autovettore associato all'unico autovalore di segno diverso da quello dei restanti tre ma basta che il vettore trovato sia "ragionevolmente" ortogonale ai vettori che rappresentano le variabili spaziali. A parte questo, se si ricerca il tempo locale in un evento "vicino" ad un altro di cui è noto il tempo locale, una buona strategia consiste nell'assegnare come vettore di tentativo (tini) questo tempo locale dell'evento vicino...

La funzione va applicata ad un tensore metrico con segnatura {+,-,-,-} e non con segnatura opposta

```
(* gn222 deve essere di ordine quattro e con elementi solo numerici *)
tempolocale[gn222_, tini_, itera_] :=
  Block[{gerschgorin, ima, bis, autova, vt, i, j, k, kk},
   (* il fattore 1.001 protegge nel caso in cui gn222 sia già diagonale *)
   gerschgorin = 1.001 Max[Sum[Abs[gn222[[k]]], {k, 1, Length[gn222]}] -
       Abs[Diagonal[gn222]] + Diagonal[gn222]];
   ima = Inverse[gerschgorin IdentityMatrix[Length[gn222]] - gn222];
   autova = Sqrt[N[tini.tini]]; vt = tini / autova;
   For[bis = -3, Abs[itera] > bis, bis++, vt = ima.vt; autova = Sqrt[N[vt.vt]];
    vt = vt / autova];
   (* qui crea la matrice unitaria di nome ima che rappresenta il risultato *)
   ima = 0 ima;
   j = 1; For[i = 1, 4 > i, i++, If[Abs[vt[[i + 1]]] > Abs[vt[[j]]], j = i + 1]];
   ima[[1]] = vt;
   For[i = 1, 4 > i, i++,
    For[kk = 0, i > kk, kk++, k = Mod[j + kk - 1, 4] + 1;
     ima[[i + 1, k]] = vt[[k]]];
    kk = Mod[k, 4] + 1;
    bis = ima[[i + 1]].ima[[i + 1]];
    ima[[i + 1]] = vt[[kk]] ima[[i + 1]];
    ima[[i + 1, kk]] = -bis;
    bis = Sqrt[ima[[i + 1]].ima[[i + 1]]];
    ima[[i + 1]] = ima[[i + 1]] / bis];
   ima
  ];
mialibreria = Append[mialibreria, "tempolocale"];
tempolocale::usage =
  "Calcola il tempo locale in un dato punto dello spazio-tempo";
```

■ Libreria TrueTensor (tt)

Raccolgo qui le funzioni della libreria che ho chiamato TrueTensor. Ogni trueTensor è una lista di almeno due elementi il primo dei quali è un normale tensore "nudo" che, in *Mathematica*, non ha la informazione del tipo di indici, mentre il secondo elemento è la lista del tipo di indici (numero pari se l'indice è covariante e dispari se è controvariante).

```
(* Crea un TrueTensor da un tensore nudo ossia senza info sul tipo di indici,
fondendolo con le info sul tipo di indici: 1→ controvariante,
2→ covariante, 0→ pseudo tensore derivato,
8→ derivata covariante, 9→derivata controvariante *)
ttensor[ma_, lcov_] := Block[{j, ld},
   If[ArrayDepth[lcov] ≠ 1, Print["ttensor:Errore 1"]; Return[False]];
   (* Se la lista dei tipi è vuota trattasi di uno scalare *)
   If[Length[lcov] == 0, Return[{ma, {}}]];
   If[ArrayDepth[ma] ≠ Length[lcov], Print["ttensor:Errore 2"]; Return[False]];
   ld = Dimensions[ma];
   If[Total[Abs[ld - Table[ld[[1]], {j, 1, Length[ld]}]]] ≠ 0,
    Print["ttensor:Errore 3"]; Return[False]];
   {ma, lcov}];
mialibreria = Append[mialibreria, "ttensor"];
ttensor::usage = "Crea un TrueTensor dato un tensore nudo";
```

```
(* contrazione di una coppia di indici. Non viene
 fatto uso del tensore metrico ma la funzione verifica solo
 che la coppia di indici contratti sia di tipo diverso *)
ttdot[ta_, na_, tb_, nb_] := Block[
   {j, n, ls, tta, itta, ttb, ittb},
   If[ ! ttensorQ[ta], Print["ttdot: non tensore il primo"]; Return[False]];
   If[ ! ttensorQ[tb], Print["ttdot: non tensore il secondo"]; Return[False]];
   n = ArrayDepth[ta[[1]]];
   If[1 > na || na > n, Print["ttdot: primo indice sconfina"]; Return[False]];
   ls = Table[j + Floor[(n + j - na) / n], {j, 1, n}];
   ls[[n]] = na;
   itta = Table[ta[[2]][[ls[[j]]]], {j, 1, n - 1}];
   ls = Table[0, {j, 1, n}]; ls[[na]] = n;
   For[k = 0; j = 1, n ≥ j, j++,
    If[ls[[j]] == 0, k++; ls[[j]] = k]];
   tta = Transpose[ta[[1]], ls];
   n = ArrayDepth[tb[[1]]];
   If[1 > nb || nb > n, Print["ttdot: secondo indice sconfina"]; Return[False]];
   ls = Table[j - 1 + Floor[(n + j - nb - 1) / n], {j, 1, n}];
   ls[[1]] = nb;
   ittb = Table[tb[[2]][[ls[[j]]]], {j, 2, n}];
   If[Mod[ta[[2]][[na]] + tb[[2]][[nb]], 2] == 0,
    Print["ttdot:attenzione : contrazione tra indici dello stesso tipo !"]];
   ls = Table[0, {j, 1, n}]; ls[[nb]] = 1;
   For[k = 1; j = 1, n ≥ j, j++,
    If[ls[[j]] == 0, k++; ls[[j]] = k]];
   ttb = Transpose[tb[[1]], ls];
   {tta.ttb, Join[itta, ittb]}];
mialibreria = Append[mialibreria, "ttdot"];
ttdot::usage = "Contrae una coppia di indici tra due true Tensor";
```

```
(* Traccia generalizzata tra due indici dello stesso
 tensore. La funzione non fa uso del tensore metrico ma si limita
 a controllare che i due indici usati siano di tipo diverso *)
ttTr[ta_, na_, nb_] := Block[
   {j, k, n, ma, mi, ls, itti, ittm},
   If[ ! ttensorQ[ta], Print["ttTr: non tensore"]; Return[False]];
   If[na == nb, Print["ttTr: indici uguali"]; Return[False]];
   ma = na; mi = nb;
   If[nb > na, ma = nb; mi = na];
   n = ArrayDepth[ta[[1]]];
   If[2 > n, Print["ttTr: rango inferiore a 2"]; Return[False]];
   If[ 1 > mi || ma > n,
    Print["ttTr: sconfinamento degli indici"]; Return[False]];
   If[Mod[ta[[2]][[ma]] + ta[[2]][[mi]], 2] == 0,
    Print["ttTr:attenzione : traccia con indici dello stesso tipo !"]];
   ls = Table[j - 1 + Floor[(n + j - mi - 1) / n], {j, 1, n}];
   ls[[1]] = mi;
   itti = Table[ta[[2]][[ls[[j]]]], {j, 1, n}];
   ls = Table[j - 1 + Floor[(n + j - ma - 1) / n], {j, 1, n}];
   ls[[1]] = ma;
   If[n == 2, ittm = {}, ittm = Table[itti[[ls[[j]]]], {j, 3, n}]];
   ls = Table[0, {j, 1, n}]; ls[[mi]] = 1;
   ls[[ma]] = 2;
   If[n > 2,
    For[k = 2; j = 1, n ≥ j, j++,
     If[ls[[j]] == 0, k++; ls[[j]] = k]]];
   {Tr[Transpose[ta[[1]], ls], Plus, 2], ittm}];
mialibreria = Append[mialibreria, "ttTr"];
ttTr::usage = "Traccia generalizzata tra due indici di un True Tensor";
```

```
(* per verificare che la variabile è
 un TrueTensor ossia è fatto come deve essere *)
ttensorQ[tt_] := Block[
   {ld},
   If[2 > Length[tt], Print["ttensorQ:err.1"]; Return[False]];
   If[ArrayDepth[tt[[2]]] ≠ 1, Print["ttensorQ:err.2"]; Return[False]];
   If[Length[tt[[2]]] == 0, Return[True]];
   If[ArrayDepth[tt[[1]]] ≠ Length[tt[[2]]],
    Print["ttensorQ:err.3"]; Return[False]];
   ld = Dimensions[tt[[1]]]; If[Total[Abs[ld - Table[ld[[1]], {j, 1, Length[ld]}]]] ≠
     0, Print["ttensorQ:err.4"]; Return[False]];
   True];
mialibreria = Append[mialibreria, "ttensorQ"];
ttensorQ::usage = "Verifica se si tratta di un True Tensor";
```

```
(* Creazione degli oggetti che servono per la metrica assegnata *)
ttmetrica[mat_, var_] := Block[{ttmet, g211, g222, ch3122, r41222, r222},
   Print["Creo la metrica"];
   g222 = {mat, {2, 2}};
   g211 = {Simplify[Inverse[g222[[1]]]], {1, 1}};
   ch3122 = {faCh3122[g222[[1]], var], {1, 0, 0}};
   r41222 = { faRiemann41222[ch3122[[1]], var], {1, 2, 2, 2}};
   r222 = { Tr[Transpose[r41222[[1]], {1, 3, 2, 4}], Plus, 2], {2, 2}};
   Print["Contenuto: {var,g222,g211,ch3122,r41222,r222}"];
   ttmet = {var, g222, g211, ch3122, r41222, r222}];
mialibreria = Append[mialibreria, "ttmetrica"];
ttmetrica::usage = "Creazione degli oggetti base di una data metrica";
```

```
(* Derivata covariante *)
ttder[ten_, metr_] := Block[{i, dbase, corre},
   If[! ListQ[ten[[1]]],
    Return[{Table[D[ten[[1]], metr[[1, i]]], {i, 1, Length[metr[[1]]]}], {8}}]
   ];
   dbase = derord[ten[[1]], metr[[1]]];
   corre = dercov[ten[[1]], metr[[4, 1]], ten[[2]]];
   {dbase + corre, Join[ten[[2]], {8}]}];
mialibreria = Append[mialibreria, "ttder"];
ttder::usage = "Derivata covariante di un True Tensor";
```

Funzioni generatrici di tensori metrici nudi

Raccolgo qui una miscellanea di funzioni, tutte inizianti con xg, usabili per creale le matrici delle metriche di uso più frequente.
Ogni oggetto creato dalla funzione è una lista di due elementi ossia la matrice e la lista delle variabili indipendenti usate.

Conclusione 2012 ... versione congelata anche se con eventuali bachi

In corso di realizzazione....

```
Sort[mialibreria]
```

```
{dercov, derord, faCh3122, faRicci222, faRiemann41222, faRiemann42222, faxt222,
 tempolocale, tralba, ttder, ttdot, ttensor, ttensorQ, ttmetrica, ttTr}
```