Libreria tensoriale di Giampaolo Bottoni (crealibtensori v.1)

Scrive su disco la libreria)

Versione (in via di sviluppo) del 22 giugno 2010.

1 Premessa

Un tensore è una lista di oggetti vari (atomici, liste o matrici).
L'ultimo elemento della lista deve essere una lista.
La lunghezza della lista dell'ultimo elemento è l'ordine del tensore + 2.
Il penultimo elemento della lista dell'ultimo elemento è un commento di qualsiasi tipo, un numero o una stringa o una lista etc.
L'ultimo elemento della lista dell'ultimo elemento è la dimensione dello spazio per il tensore considerato.
I precedenti elementi indicano la natura degli indici del tensore ossia se sono controvarianti (1), covarianti (2), derivativi ordinari(0), derivati covarianti covarianti (8), derivati covarianti controvarianti (9), personalizzati (altre cifre escluse 0,1,2,8,9).

Con questa definizione è possibile trattare come tensore anche uno scalare che ha questa struttura:

[valore,[commento,dimensione_spazio]]

La dimensione_spazio minima ammessa è 2 mentre per descrivere lo spazio tridimensionale deve essere:

dimensione_spazio:3

Per descrivere gli spazi relativistici pseudoeuclidei ossia con la prima dimensione temporale e le restanti tre spaziali deve essere:

dimensione_spazio:4

I tensori, date le scelte di programmazione ora
fatte, si distinguono tra tensori con un numero
dispari di indici e tensori con un numero pari
di indici.
Se il tensore ha numero di indici pari viene usato
solo il primo elemento del tensore mentre se
il numero è dispari, vengono usati i primi elementi
da 1 a dimensione_spazio.

Un vettore è un tensore di ordine 1 di questa
struttura, nel caso in cui la dimensione
dello spazio sia 2:

[v1,v2,[i1,commento,2]]

Se la dimensione dello spazio vale 3 il vettore
ha questa struttura:

[v1,v2,v3,[i1,commento,3]]

Se la dimensione dello spazio vale 4 il vettore
ha questa struttura:

[v1,v2,v3,v4,[i1,commento,4]]

Il valore di i1 è 1 se il vettore è controvariante
ed è 2 se è covariante etc.

1.1 Operazioni di salvataggio su file

Specifico il file in cui scrivo la nuova libreria:

```
(%i1) salvoqui:"c:/xmaxima/libtensori.mc";
(%o1) c:/xmaxima/libtensori.mc
```

```
(%i2) if atom(path_iniziale) then writefile(salvoqui);
Starts dribbling to c:/xmaxima/libtensori.mc (2010/6/22, 22:51:54).
NIL
(%o2) done
```

```
(%i3) print(["Ora comincio a lavorare su ",salvoqui])$
[Ora comincio a lavorare su ,c:/xmaxima/libtensori.mc]
```

```
(%i4) cartellalavoro:"C:/xmaxima/###.{mc,mac}";
(%o4) C:/xmaxima/###.{mc,mac}
```

E' consigliabile, prima di modificarlo, di salvare il valore
del path di default di Maxima.

```
(%i5) ( if atom(path_iniziale) then
          (path_iniziale: file_search_maxima) )$
```

```
(%i6) file_search_maxima: cons(cartellalavoro,path_iniziale);
(%o6) [C:/xmaxima/###.{mc,mac},C:/Users/Giampaolo/maxima/###.{mac,mc},
C:\PROGRA~2\MAXIMA~1.0/share/maxima/5.21.0/share/###.{mac,mc},
C:\PROGRA~2\MAXIMA~1.0/share/maxima/5.21.0/share/{affine,algebra,algebra/charse
]
```

Amplio il path iniziale aggiungendogli una cartella. Con questo trucco posso ricaricare varie volte questo documento senza il problema di modificare ogni volta il path di ricerca.

```
(%i7) file_search_maxima: cons(cartellalavoro,path_iniziale);
(%o7) [C:/xmaxima/###.{mc,mac},C:/Users/Giampaolo/maxima/###.{mac,mc},
C:\PROGRA~2\MAXIMA~1.0/share/maxima/5.21.0/share/###.{mac,mc},
C:\PROGRA~2\MAXIMA~1.0/share/maxima/5.21.0/share/{affine,algebra,algebra/charse
]
```

```
(%i8) print(["Il path usato è il seguente",cartellalavoro]);
[Il path usato è il seguente,C:/xmaxima/###.{mc,mac}]
(%o8) [Il path usato è il seguente,C:/xmaxima/###.{mc,mac}]
```

2 Funzioni algebriche

2.1 Funzioni di informazione

L'ordine di un tensore è ottenibile con la seguente funzione che, per ragioni di velocità, non effettua nessuna verifica della correttezza del tensore tn passato in argomento.

```
(%i9) ordine(tn):=block([],
          length(tn[length(tn)])-2
          )$
```

```
(%i10) libmia:["ordine(tn)"];
(%o10) [ordine(tn)]
```

Il commento che ogni tensore contiene e che può essere una stringa ma anche un numero o una lista o un qualsiasi oggetto, è estraibile in questo modo:

```
(%i11) commento(tn):=block([u],
          u:length(tn),
          tn[u][length(tn[u])-1])$
```

```
(%i12) libmia:cons("commento(tn)",libmia);
(%o12) [commento(tn),ordine(tn)]
```

Per inserire un commento nel tensore, uso la commentalo:

```
(%i13) commentalo(tn,comme):=block([u],
           u:length(tn),
           tn[u][length(tn[u])-1]:comme,tn[u] )$

(%i14) libmia:cons("commentalo(tn,comme)",libmia);
(%o14) [commentalo(tn,comme),commento(tn),ordine(tn)]
```

Ogni tensore contiene, come si è detto, la dimensione
dello spazio a cui si riferisce. Per ottenere questo
dato si può usare questa funzione:

```
(%i15) dimensione(tn):=block([],
           tn[length(tn)][length(tn[length(tn)])])$

(%i16) libmia:cons("dimensione(tn)",libmia);
(%o16) [dimensione(tn),commentalo(tn,comme),commento(tn),ordine(tn)]
```

Per estrarre dal tensore la lista dei tipi (covarianti,
controvarianti, etc.) dei suoi indici si può usare
questa funzione:

```
(%i17) indici(tn):=block([],
           rest(tn[length(tn)],-2))$

(%i18) libmia:cons("indici(tn)",libmia)$
```

Indichiamo con tn un generico tensore ed esaminamo
i test che ogni funzione, quando non ci sono
esigenze di prestazioni, potrebbe fare per
controllare che la struttura del tensore è corretta.
Raccogliamo i test nella funzione tensorp(tn)
che dà risultato true se tn è un tensore
ben fatto e false se tn non è un tensore o è un
tensore mal fatto.
La tensorp utilizza in modo ricorsivo la
matrixxp che opera su matrici di matrici.

```
(%i19) matrixxp(tn,o,n):=block([basta,smetti],
          basta:false,smetti:true,
          if o>1 then (
             smetti:false,
             if not(matrixp(tn)) then basta:true,
             if length(tn[1])#n then basta:true
             ),
          if basta then return(false),
          if smetti then return(true),
          matrixxp(tn[1,1],o-2,n)
          )$
```

```
(%i20) tensorp(tn):=block([u,o,n,pd],
          if not(listp(tn)) then return(false),
          u:length(tn),
          if not(listp(tn[u])) then return(false),
          o:length(tn[u])-2,
          if 0>o then return(false),
          n:floor(tn[u][o+2]),
          if 2>n then return(false),
          pd:mod(o,2),
          if pd=0 then ( if u>n then return(false))
          else if n>u-1 then return(false),
          if matrixxp(tn[1],o,n) then return(true),
          false )$
```

```
(%i21) libmia:cons("tensorp(tn)",libmia)$
```

2.2 zerotensor: funzione di inizializzazione

La funzione zerotensor genera un tensore con elementi
tutti nulli, con indici di valore prescelto,
con commento arbitrario e di dimensione assegnata.

```
(%i22) zerotensor(tipi,comme,ndim):=block([tn,j,o,u,nd],
          if not(listp(tipi)) then
             return("Errore, non lista tipi"),
          o:length(tipi), nd:floor(ndim),
          if 2>nd then
             return("Errore, dimensione inferiore a 2"),
          if o=0 then return([0,[comme,nd]]),
          if mod(o,2)=0 then tn:makelist(0,j,1,2)
          else tn:makelist(0,j,1,nd+1),
          u:length(tn),
          tn[u]:append(makelist(tipi[j],j,1,o),[comme,nd]),
          for j:1 thru u-1 do tn[j]:zeromatmat(o,nd),
          tn
          )$
```

```
(%i23) zeromatmat(o,nd):=block([tr,omm,j,k],
          if 2>o then return(0),
          tr:zeromatrix(nd,nd),
          omm:o-2,
          for j:1 thru nd do
          for k:1 thru nd do
              tr[j,k]:zeromatmat(omm,nd),
          tr )$
```

```
(%i24) libmia:cons("zerotensor(tipi,comme,ndim)",libmia)$
```

2.3 Funzioni per generare tensori da scalari, liste o matrici

Per metrica si intende una lista di tensori utili
per gestire una data metrica.
La lista viene generata da una apposita funzione
fa_metrica(matrice,listavariabili) definita
in qui in seguito.

Crea un tensore scalare

```
(%i25) tsca(sc,metrica):=block([],
           [sc,["(scal)",dimensione(metrica[2])]])$
```

```
(%i26) libmia:cons("tsca(sc,metrica)",libmia)$
```

Crea un vettore covariante, data una lista e
la metrica.

```
(%i27) tcov(lv,metrica):=block([nl,nd],
           if not(listp(lv)) then
              return("Non lista"),
           nl:length(lv),
           nd:dimensione(metrica[2]),
           [ makelist(lv[1+mod(j-1,nl)],j,1,nd),
             [2,"(v2)",nd]])$
```

```
(%i28) libmia:cons("tcov(lv,metrica)",libmia)$
```

Crea un vettore controvariante, data la lista e
la metrica.

```
(%i29) tcontrov(lv,metrica):=block([nl,nd],
           if not(listp(lv)) then
              return("Non lista"),
           nl:length(lv),
           nd:dimensione(metrica[2]),
           [ makelist(lv[1+mod(j-1,nl)],j,1,nd),
             [1,"(v1)",nd]])$

(%i30) libmia:cons("tcontrov(lv,metrica)",libmia)$

Crea un tensore di ordine 2, totalmente controvariante,
data una matrice e la metrica.

(%i31) tmat11(mat,metrica):=block([],
           if not(matrixp(mat)) then
              return("Non matrice"),
           [mat,[1,1,"(tm11)",dimensione(metrica[2])]])$

(%i32) libmia:cons("tmat11(mat,metrica)",libmia)$

Crea un tensore di ordine 2, misto controvariante,
covariante, data una matrice e la metrica.

(%i33) tmat12(mat,metrica):=block([],
           if not(matrixp(mat)) then
              return("Non matrice"),
           [mat,[1,2,"(tm12)",dimensione(metrica[2])]])$

(%i34) libmia:cons("tmat12(mat,metrica)",libmia)$

Crea un tensore di ordine 2, misto covariante,
controvariante, data una matrice e la metrica.

(%i35) tmat21(mat,metrica):=block([],
           if not(matrixp(mat)) then
              return("Non matrice"),
           [mat,[2,1,"(tm11)",dimensione(metrica[2])]])$

(%i36) libmia:cons("tmat21(mat,metrica)",libmia)$

Crea un tensore di ordine 2, totalmente covariante,
data una matrice e la metrica.

(%i37) tmat22(mat,metrica):=block([],
           if not(matrixp(mat)) then
              return("Non matrice"),
           [mat,[2,2,"(tm22)",dimensione(metrica[2])]])$
```

```
(%i38) libmia:cons("tmat22(mat,metrica)",libmia)$
```

2.4 Operazioni di somma, differenza e prodotto per scalare, tra tensori.

Somma di due tensori con uguale dimensione, ordine
e tipo di indici.

```
(%i39) tsomma(tena,tenb):=block(
          [oa,ob,na,nb,ina,inb,stop],
          if not(tensorp(tena)) then
             return("Errore: primo arg. non tensore"),
          if not(tensorp(tenb)) then
             return("Errore: secondo arg. non tensore"),
          oa:ordine(tena), ob:ordine(tenb),
          if oa#ob then
             return("Errore: tensori di ordine diverso"),
          na:dimensione(tena),  nb:dimensione(tenb),
          if na#nb then
             return("Errore: tensori di dimensione diversa"),
          ina:indici(tena), inb:indici(tenb),
          stop:false,
          for j:1 thru oa do if mod(ina[j]+inb[j],2)=1
              then stop:true,
          if stop then
             return("Errore: indici incongruenti"),
          append(ratsimp(rest(tena,-1)+rest(tenb,-1)),
             [append(ina,["(Somma tensori)",na])])
          )$
```

```
(%i40) libmia:cons("tsomma(tena,tenb)",libmia)$
```

Differenza tra due tensori con uguale dimensione, ordine
e tipo di indici.

```
(%i41) tmeno(tena,tenb):=block(
          [oa,ob,na,nb,ina,inb,stop],
          if not(tensorp(tena)) then
             return("Errore: primo arg. non tensore"),
          if not(tensorp(tenb)) then
             return("Errore: secondo arg. non tensore"),
          oa:ordine(tena), ob:ordine(tenb),
          if oa#ob then
             return("Errore: tensori di ordine diverso"),
          na:dimensione(tena),  nb:dimensione(tenb),
          if na#nb then
             return("Errore: tensori di dimensione diversa"),
          ina:indici(tena), inb:indici(tenb),
          stop:false,
          for j:1 thru oa do if mod(ina[j]+inb[j],2)=1
              then stop:true,
          if stop then
             return("Errore: indici incongruenti"),
          append(ratsimp(rest(tena,-1)-rest(tenb,-1)),
             [append(ina,["(Differenza tra tensori)",na])])
          )$
```

```
(%i42) libmia:cons("tmeno(tena,tenb)",libmia)$
```

Prodotto di un tensore scalare per un tensore della stessa dimensione.

```
(%i43) tprod(tscala,tenb):=block(
          [oa,ob,na,nb,inb],
          if not(tensorp(tscala)) then
             return("Errore: primo arg. non tensore"),
          if not(tensorp(tenb)) then
             return("Errore: secondo arg. non tensore"),
          oa:ordine(tscala), ob:ordine(tenb),
          if oa#0 then
             return("Errore: non scalare"),
          na:dimensione(tscala),  nb:dimensione(tenb),
          if na#nb then
             return("Errore: tensori di dimensione diversa"),
          inb:indici(tenb),
          append(ratsimp(tscala[1]*rest(tenb,-1)),[append(indici(tenb),
             ["(Prodotto scalare*tensore)",nb])])
          )$
```

```
(%i44) libmia:cons("tprod(tscala,tenb)",libmia)$
```

2.5 tassegna e tvale : per assegnare o conoscere valori di singoli elementi.

Per non dovere specificare gli indici accoppiandoli.
Assegna un dato ad un elemento di matrice di
matrice oppure trova il valore di quell'elemento.
Sono funzioni ricorsive:

```
(%i45) assegnam(mt,o,val,p):=block([],
           if o=length(p) then (
              mt[p[o-1],p[o]]:val,
              return(true)),
           assegnam(mt[p[o-1],p[o]],o+2,val,p)
           )$
```

```
(%i46) valem(miot,o,p):=block([],
           if o=length(p) then return(miot[p[o-1],p[o]]),
           valem(miot[p[o-1],p[o]],o+2,p)
           )$
```

Assegna un valore ad un elemento del tensore oppure
trova il valore di quell'elemento.

```
(%i47) tassegna(tn,val,p):=block([o],
           o:length(tn[length(tn)])-2,
           if length(p)#o then
           return("errore: incongruenza di indici"),
           if o=0 then ( tn[1]:val,return(true)),
           if o=1 then ( tn[p[1]]:val, return(true)),
           if mod(o,2)=0 then assegnam(tn[1],2,val,p)
           else assegnam(tn[p[1]],2,val,rest(p))
           )$
```

```
(%i48) tvale(tn,p):=block([o],
           o:length(tn[length(tn)])-2,
           if length(p)#o then
           return("errore: incongruenza di indici"),
           if o=0 then return(tn[1]),
           if o=1 then return(tn[p[1]]),
           if mod(o,2)=0 then valem(tn[1],2,p)
           else (
               valem(tn[p[1]],2,rest(p)))
           )$
```

```
(%i49) libmia:cons("tassegna(tn,val,p)",libmia)$
```

```
(%i50) libmia:cons("tvale(tn,p)",libmia)$
```

2.6 permuta : permutazione di indici di un tensore

La funzione che permuta gli indici di un qualsiasi
tensore creando un altro tensore con gli indici permutati
in base all'ordinamento specificato dalla lista per.
La lista per deve indicare da dove è stato preso il dato
che compare in quella posizione.
Per esempio [3,1,2,4] indica che il dato che compare
come primo indice è quello che prima era situato in
colonna 3, come secondo dato quello che prima stava in
colonna 1 etc.
La rotazione degli indici sa fa con [2,3,4,1].

```
(%i51) permuta(tn,per,comm):=block([tr,u,o,a,b,j,nd,va,vb,id],
           if not(tensorp(tn))
              then return("Errore, non tensore"),
           if not(listp(per)) then
              return("Errore, non lista permutazioni"),
           u:length(tn),o:length(tn[u])-2,
           if 2>o then return("Applicabile da ordine 2 in su"),
           if o#length(per) then
              return("Errore, lista permutazioni incongruente"),
           j:1,for a:1 thru o-1 do for b:a+1 thru o do
               if per[a]=per[b] then j:0,
           if j=0 then return("Permutazione errata"),
           nd:tn[u][o+2],
           va:makelist(0,j,1,o),
           vb:makelist(0,j,1,o),
           id:makelist(0,j,1,o),
           for j:1 thru o do id[j]:tn[u][per[j]],
           tr:zerotensor(id,comm,nd),
           funfor(tr,tn,va,vb,per,1,o,nd),
           tr
           )$
```

Faccio un ciclo for in forma ricorsiva...

```
(%i52) funfor(ta,tb,va,vb,per,n,m,nd):=block([j,np],
          if n=m then (
             for j:1 thru nd do (
                va[n]:j,
                vb[per[n]]:j,
                tassegna(ta,tvale(tb,vb),va) ))
          else (
             np:n+1,
             for j:1 thru nd do (
                 va[n]:j,
                 vb[per[n]]:j,
                 funfor(ta,tb,va,vb,per,np,m,nd))
                )
          )$
```

```
(%i53) libmia:cons("permuta(tn,per,com)",libmia)$
```

Funzione per trovare come è permutata una lista ossia
data una lista e una permutazione, cosa diventa la
lista permutata.

```
(%i54) ada(la,per):=block([n,j,k,lr],
           n:length(per),
           if n#length(la) then return("Disuguale lunghezza liste"),
           for j:1 thru n-1 do for k:j+1 thru n do if
              per[j]=per[k] then return("Indici duplicati"),
           lr:makelist(0,j,1,n),
           for j:1 thru n do lr[j]:la[per[j]],
           lr)$
```

```
(%i55) libmia:cons("ada(la,per)",libmia)$
```

2.7 tensoredalista : per creare un tensore usando una lista per inizializzarlo

Crea un tensore del tipo voluto, prelevando i dati da una lista
che se è troppo breve viene riutilizzata ciclicamente e
naturalmente, se è troppo lunga, viene usata solo parzialmente.

```
(%i56) forlista(tb,vb,n,m,nd,lista,dove):=block([j,np],
          if n=m then (
             for j:1 thru nd do (
                vb[n]:j,
                dove[1]:mod(dove[1],length(lista))+1,
                tassegna(tb,lista[dove[1]],vb) ))
          else (
             np:n+1,
             for j:1 thru nd do (
                 vb[n]:j,
                 forlista(tb,vb,np,m,nd,lista,dove))
             )
          )$
```

```
(%i57) tensoredalista(lista,tipi,comme,ndim):=block(
           [tr,dove,m,vb,j],
           tr:zerotensor(tipi,comme,ndim),
           dove[1]:0,
           m:length(tipi),
           vb:makelist(0,j,1,m),
           forlista(tr,vb,1,m,ndim,lista,dove),
           tr)$
```

```
(%i58) libmia:cons("tensoredalista(lista,tipi,comme,ndim)",libmia)$
```

2.8 listadatensore : per estrarre gli elementi di un tensore mettendoli in una lista

Estrae gli elementi di un tensore producendo una lista.

```
(%i59) forinlista(lista,tb,vb,n,m,nd,dove):=block([j,np],
          if n=m then (
             for j:1 thru nd do (
                vb[n]:j,
                dove[1]:mod(dove[1],length(lista))+1,
                lista[dove[1]]:tvale(tb,vb)))
          else (
             np:n+1,
             for j:1 thru nd do (
                vb[n]:j,
                forinlista(lista,tb,vb,np,m,nd,dove))
                )
          )$
```

```
(%i60) listadatensore(tn):=block([lista,u,o,dove,vb,j],
           u:length(tn),o:length(tn[u])-2,
           nd:tn[u][o+2],
           dove[1]:0,
           vb:makelist(0,j,1,o),
           lista:makelist(0,j,1,nd^o),
           forinlista(lista,tn,vb,1,o,nd,dove),
           lista)$
```

```
(%i61) libmia:cons("listadatensore(tn)",libmia)$
```

2.9 traccia : generalizzazione del calcolo della traccia di una matrice.

traccia di un tensore, generalizzazione di quella di una matrice...

```
(%i62) fortraccia(tr,vr,tn,vn,h,k,n,nn,m,nd):=block([j,np,nnp,ss],
          if n=m then (
             ss:0,
             for j:1 thru nd do (
                vn[h]:j,vn[k]:j,
                ss:ss+tvale(tn,vn)),
                tassegna(tr,ss,vr))
          else (
             np:n+1, nnp:nn+1,
             if nnp=h then nnp:nnp+1,
             if nnp=k then nnp:nnp+1,
             for j:1 thru nd do (
                vr[np]:j,
                vn[nnp]:j,
                fortraccia(tr,vr,tn,vn,h,k,np,nnp,m,nd))
                )
          )$
```

```
(%i63) traccia(tn,ih,ik):=block([u,o,nd,vr,vb,h,k,tipi,i,j],
           if not(tensorp(tn)) then
              return("Errore: non tensore"),
           u:length(tn),o:length(tn[u])-2,nd:tn[u][o+2],
           if 2>o then return("Errore, ordine inferiore a 2"),
           if ih=ik then return("Errore, indici uguali"),
           h:min(ih,ik),k:max(ih,ik),
           if k>o then
              return(["Errore, indice superiore ad ordine",k,o]),
           vb:makelist(0,j,1,o),vr:[],
           if o>2 then vr:makelist(0,j,1,o-2),
           tipi:indici(tn),
           if tipi[h]=tipi[k] then
               print("Attenzione: indici dello stesso tipo"),
           tr:zerotensor(vr,"ridotto",nd),
           i:0,for j:1 thru o do (
               if j#h then if j#k then (
                  i:i+1,
                  tr[u][i]:tipi[j])),
           fortraccia(tr,vr,tn,vb,h,k,0,0,o-2,nd),
           tr)$
```

```
(%i64) libmia:cons("traccia(tn,ih,ik)",libmia)$
```

2.10 scala : generalizzazione del prodotto scalare tra vettori e tra matrici.

```
Prodotto scalare generalizzato. Attenzione... è posizionale
e per ottenere gli indici nell'ordine voluto bisogna poi
riordinare il risultato con la funzione permuta.
```

```
(%i65) forscala(tr,vr,ta,va,tb,vb,h,k,n,nn,ma,m,nd):=block(
          [j,np,nnp,ss],
          if n=m then (
             va[h]:1,
             vb[k]:1,
             ss:0,
             for j:1 thru nd do (
                va[h]:j,vb[k]:j,
                ss:ss+tvale(ta,va)*tvale(tb,vb)),
                tassegna(tr,ss,vr))
          else (
             np:n+1, nnp:nn+1,
             if nnp=h then nnp:nnp+1,
             if nnp=k+ma then nnp:nnp+1,
             for j:1 thru nd do (
                vr[np]:j,
                if nnp>ma then vb[nnp-ma]:j
                else va[nnp]:j,
                forscala(tr,vr,ta,va,tb,vb,h,k,np,nnp,ma,m,nd))
                )
          )$
```

```
(%i66) scala(ta,ia,tb,ib):=block(
          [ua,oa,ub,ob,ur,ot,nd,vr,va,vb,tipa,tipb,i,j],
          if not(tensorp(ta)) then
             return("Errore: ta non tensore"),
          if not(tensorp(tb)) then
             return("Errore: tb non tensore"),
          ua:length(ta),oa:length(ta[ua])-2,nd:ta[ua][oa+2],
          if 1>oa then return("Errore, ordine ta inferiore a 1"),
          ub:length(tb),ob:length(tb[ub])-2,
          if nd#tb[ub][ob+2] then
             return("Errore, dimensioni diverse"),
          if 1>ob then return("Errore, ordine tb inferiore a 1"),
          if ia>oa then
             return(["Errore, indice superiore ad ordine",ia,oa]),
          if ib>ob then
             return(["Errore, indice superiore ad ordine",ib,ob]),
          va:makelist(0,j,1,oa),vb:makelist(0,j,1,ob),
          ot:oa+ob-2,
          vr:[],if ot>0 then vr:makelist(0,j,1,ot),
          tipa:indici(ta),tipb:indici(tb),
          if tipa[ia]=tipb[ib] then
              print("Attenzione: indici dello stesso tipo"),
          tr:zerotensor(vr,"prod-scalare",nd),
          ur:length(tr),
          i:0,for j:1 thru oa+ob do (
             if j>oa then ( if (j-oa)#ib then (
                i:i+1,
                tr[ur][i]:tipb[j-oa]))
             else if j#ia then (
                i:i+1,
                tr[ur][i]:tipa[j])),
          forscala(tr,vr,ta,va,tb,vb,ia,ib,0,0,oa,ot,nd),
          tr)$
```

```
(%i67) libmia:cons("scala(ta,ia,tb,ib)",libmia)$
```

3 Funzioni per il calcolo differenziale

3.1 fa_diff : derivata ordinaria di un tensore

Richiede l'uso di una permutazione finale perché spontaneamente aggiungerebbe l'indice 0 all'inizio e non alla fine.

```
(%i68) fa_diff(tn,variabili,comme):=block([tn0,tnn,indi,o,j],
           tnn:rest(tn,-1),
           o:ordine(tn),
           indi:indici(tn),
           nd:dimensione(tn),
           if mod(o,2)=0 then (
              tn0:makelist(0,j,1,nd+1),
              for j:1 thru nd do
              tn0[j]:ratsimp(diff(tnn[1],variabili[j])),
              tn0[nd+1]:append(cons(0,indi),[comme,nd])
              )
           else (
              tn0:[0,append(cons(0,indi),[comme,nd])],
              tn0[1]:ratsimp(matrix(diff(tnn,variabili[1]))),
              for j:2 thru nd do
              tn0[1]:addrow(tn0[1],
                     ratsimp(matrix(diff(tnn,variabili[j]))))
              ),
           if o>0 then
              permuta(tn0,makelist(mod(j,o+1)+1,j,1,o+1),comme)
           else tn0
           )$
```

```
(%i69) libmia:cons("fa_diff(tn,variabili,comme)",libmia)$
```

3.2 fa_ch222 : Simboli di Christoffel di prima specie

```
(%i70) fa_ch222(g220):=block([g022,g202],
            if controllo then print("Inizia fa_ch222"),
            g022:permuta(g220,[2,3,1],"-"),
            g202:permuta(g022,[2,3,1],"-"),
            append((+rest(g220,-1)+rest(g022,-1)-rest(g202,-1))/2,
            [[2,2,2,"Christoffel prima specie",dimensione(g220)]])
            )$
```

```
(%i71) libmia:cons("fa_ch222(g220)",libmia)$
```

3.3 fa_ch122 : Per fare i simboli di Christoffel di seconda specie

```
Eleva il primo indice.. e fa il simbolo di Christoffel
di seconda specie. Anche in questo caso il secondo e
il terzo indice sono scambiabili.
```

```
(%i72) fa_ch122(g22,listavariabili):=block([g11,g220],
         if controllo then print("Inizia fa_ch222"),
         g11: [ratsimp(invert(g22[1])),
               [1,1,"-",dimensione(g22)]],
         g220: fa_diff(g22,listavariabili,"g220"),
         ratsimp(scala(g11,2,fa_ch222(g220),1))
         )$
```

```
(%i73) libmia:cons("fa_ch122(g22,listavariabili)",libmia)$
```

3.4 fa_r1222 : genera il tensore di Riemann col primo indice controvariante.

```
Ora calcolo il tensore di Riemann con la formula dell'HEL
Hobson,Efstathiou,Lasenby ISBN 9780521829519 a pag. 158.
Ricopio la formula dal libro:
r1d2a2b2c = ch1d2a2a0b -
            ch1d2a2b0c +
            ch1e2a2c*ch1d2e2b -
            ch1e2a2b*ch1d2e2c
Il nome degli indici però non mi piace e preferisco usare
come nomi a,b,c,d sostituti di d,a,b,c ed inoltre usare
il nome j al posto del nome e, per cui la formula diventa:
r1a2b2c2d = ch1a2b2d0c -
            ch1a2b2c0d +
            ch1j2b2d*ch1a2j2c -
            ch1j2b2c*ch1a2j2d
In linguaggio Maximese non interpongo i nomi degli indici
alle cifre indicatrici di tipo ma scrivo i nomi tra
parentesi quadre:
r1222[a,b][c,d]:ch1220[a,b][d,c]-
                ch1220[a,b][c,d]+
                ch122[j][b,d]*ch122[a][j,c]-
                ch122[j][b,c]*ch122[a][j,d]
Per fare il calcolo bisogna disporre sia del simbolo di
Christoffel di seconda specie che dello pseudotensore
delle sue derivate.
Spezzo il calcolo in due parti in modo che sia possibile
partire o dal tensore metrico covariante o dai simboli
di Christoffel di seconda specie.
```

```
(%i74) fa_r1222(t22,listavariabili):=block(
          [nd,g22,g11,g220,g022,g202,ch222,ch122],
          if controllo then print("Inizia fa_r1222"),
          if not(tensorp(t22)) then
             return("Primo argomento non tensore"),
          nd:dimensione(t22),
          g22:[ratsimp((t22[1]+transpose(t22[1]))/2),[2,2,"(g22)",nd]],
          g11:[ratsimp(invert(g22[1])),[1,1,"(g11)",nd]],
          g220:fa_diff(g22,listavariabili,"(g220)"),
          g022:permuta(g220,[2,3,1],"-"),
          g202:permuta(g022,[2,3,1],"-"),
          ch222:append((rest(g220,-1)+rest(g022,-1)-rest(g202,-1))/2,
                [[2,2,2,"Christoffel prima specie",dimensione(g220)]]),
          ch122:scala(g11,2,ch222,1),
          fa_ch_r1222(ch122,listavariabili))$

(%i75) fa_ch_r1222(ch122,listavariabili):=block(
          [r1222,ch1a2b2c2d,ch1a2b2d2c,xabcd,xacbd,xadbc],
          if controllo then print("Inizia fa_ch_r1222"),
          ch1a2b2c2d:fa_diff(ch122,listavariabili,"ch1a2b2c2d"),
          ch1a2b2d2c:permuta(ch1a2b2c2d,[1,2,4,3],"ch1a2b2d2c"),
          xabcd:scala(ch122,3,ch122,1),
          xacbd:permuta(xabcd,[1,3,2,4],"xacbd"),
          xadbc:permuta(xabcd,[1,4,3,2],"xadbc"),
          r1222:[ch1a2b2d2c[1]-ch1a2b2c2d[1]+ xacbd[1] - xadbc[1],
                [1,2,2,2,"(r1222)",dimensione(ch122)]]
          )$

(%i76) libmia:cons("fa_r1222(t22,listavariabili)",libmia)$

VERIFICA:
Le funzioni della libreria alternativa, usabile
per calcolare il tensore di Riemann r1222 sono queste
che non sono ricorsive e non gestiscono gli indici.

(%i77) der2(t2,variabili):=block([t3,nd,j],
          nd:length(variabili),
          t3:makelist(0,j,1,nd),
          for j:1 thru nd do
              t3[j]:diff(t2,variabili[j]),
          t3)$

(%i78) chris222(g22,variabili):=block([g220,ss,tn],
          g220:r3(der2(g22,variabili)),
          ss:r3(g220),
          tn:g220+ss,
          ss:r3(ss),
          ratsimp((tn-ss)/2)
       )$
```

```
(%i79) chris122(g22,variabili):=block([g11],
       g11:invert(g22),
       ch222:chris222(g22,variabili),
       transpose(g11.ch222)[1]
       )$
```

```
(%i80) r3(t3):=block([nd,j,k,h,t3n],
        nd:length(t3),
        t3n:makelist(ident(nd),j,1,nd),
        for j:1 thru nd do
        for k:1 thru nd do
        for h:1 thru nd do
          t3n[j][k,h]:t3[h][j,k],
        t3n
       )$
```

```
(%i81) r4(t4):=block([nd,i,j,k,h,t4n],
       nd:length(t4),
       t4n:genmatrix(lambda([i,j],ident(nd)),nd,nd),
       for i:1 thru nd do
       for j:1 thru nd do
       for k:1 thru nd do
       for h:1 thru nd do
          t4n[i,j][k,h]:t4[h,i][j,k],
       t4n
       )$
```

```
(%i82) der3(t3,variabili):=block([t4,t4d,nd,j],
          if not(listp(t3)) then (print(["Errore",t3]),
          return("Non lista t3")),
          nd:length(variabili),
          t4d:der2(t3,variabili),
          t4:matrix(t4d[1]),
          for j:2 thru nd do
            t4:addrow(t4,t4d[j]),
          ratsimp(t4))$
```

```
Bisogna chiamare la riemann_1222 passandogli il tensore metrico
covariante e lei fa tutto...
Questa funzione è utile perché è più semplice di quella
ricorsiva ed adatta ad essere riempita di stampe di controllo...
```

```
(%i83) riemann_1222(t22,variabili):=block(
          [g22,ch122,ch1220,r1222,nd,a,b,c,d,j,sp],
          if not(tensorp(t22)) then
             return("Primo argomento non tensore"),
          nd:dimensione(t22),
          g22:ratsimp((t22[1]+transpose(t22[1]))/2),
          ch122:chris122(g22,variabili),
          ch1220:r4(der3(ch122,variabili)),
          r1222:genmatrix(lambda([i,j],ident(nd)),nd,nd),
          for a:1 thru nd do
          for b:1 thru nd do
          for c:1 thru nd do
          for d:1 thru nd do (
             sp:ch1220[a,b][d,c]-ch1220[a,b][c,d],
             for j:1 thru nd do sp:sp+
                  ch122[j][b,d]*ch122[a][j,c]-
                  ch122[j][b,c]*ch122[a][j,d],
             r1222[a,b][c,d]:sp),
          [ ratsimp(r1222),[1,2,2,2,"(r1222)",nd]]
          )$

(%i84) libmia:cons("riemann_1222(t22,variabili)",libmia)$
```

3.5 fa_r2222 : genera il tensore di Riemann totalmente covariante.

Calcolo il tensore di Riemann in forma totalmente
covariante, in base a quanto sta scritto sull' HEL.

```
r2a2b2c2d = (g2b2c0d0a-
             g2a2c0d0b+
             g2a2d0b0c-
             g2b2d0a0c )/2 +
             ch1j2b2c*ch2j2a2d -
             ch1j2b2d*ch2j2a2c
```

ovvero in Maximese

```
r2222[a,b][c,d] = ( g2200[b,c][a,d] -
                    g2200[a,c][b,d] +
                    g2200[a,d][b,c] -
                    g2200[b,d][a,c] )/2 +
                    ch122[j][a,d]*ch222[j][b,c] -
                    ch122[j][a,c]*ch222[j][b,d]
```

Il tensore di Riemann in forma totalmente covariante
evidenzia le sue simmetrie. E' antisimmetrico scambiando
gli indici della prima coppia ed e' antisimmetrico
scambiando gli indici della seconda coppia mentre
è simmetrico scambiando le coppie tra loro.
Per queste simmetrie il calcolo del tensore di Riemann
potrebbe essere molto più veloce evitando inutili ricalcoli
ma qui ...non si bada a spese...

```
(%i85) fa_r2222(t22,listavariabili):=block(
         [nd,g22,xg11,g11,g220,g022,g202,ch222,ch122,g2200],
         if controllo then print("Inizia fa_r2222"),
         if not(tensorp(t22)) then
            return("Primo argomento non tensore"),
         nd:dimensione(t22),
         g22:[ratsimp((t22[1]+transpose(t22[1]))/2),[2,2,"(g22)",nd]],
         xg11:trigsimp(ratsimp(invert(g22[1]))),
         g11:[xg11,[1,1,"(g11)",nd]],
         g220:fa_diff(g22,listavariabili,"(g220)"),
         g022:permuta(g220,[2,3,1],"-"),
         g202:permuta(g022,[2,3,1],"-"),
         ch222:append(ratsimp((rest(g220,-1)+
               rest(g022,-1)-rest(g202,-1))/2),
               [[2,2,2,"Christoffel prima specie",dimensione(g220)]]),
         ch122:scala(g11,2,ch222,1),
         g2200:fa_diff(g220,listavariabili,"(yabcd) ovvero g2200"),
         fa_poi_r2222(ch222,ch122,g2200)
              )$
```

```
(%i86) fa_poi_r2222(ch222,ch122,g2200):=block(
          [zabcd,zadbc,zacbd,ybcad,yacbd,yadbc,ybdac],
          if controllo then print("Inizia fa_poi_r2222"),
          zabcd:scala(ch122,1,ch222,1),
          zadbc:permuta(zabcd,[1,3,4,2],"(zadbc)"),
          zacbd:permuta(zabcd,[1,3,2,4],"(zacbd)"),
          ybcad:permuta(g2200,[3,1,2,4],"(ybcad)"),
          yacbd:permuta(g2200,[1,3,2,4],"(yacbd)"),
          yadbc:permuta(g2200,[1,3,4,2],"(yadbc)"),
          ybdac:permuta(g2200,[3,1,4,2],"(ybdac)"),
          [ratsimp((ybcad[1]-yacbd[1]+yadbc[1]-ybdac[1])/2
                    +zadbc[1]-zacbd[1]),
            [2,2,2,2,"(r2222)",nd]] )$

(%i87) libmia:cons("fa_r2222(t22,listavariabili)",libmia)$

Ecco la funzione tratta della altra libreria...
Sembra piu' veloce essendo meno generale dato che non fa uso
di funzioni ricorsive ma utilizza il costrutto for.

(%i88) riemann_2222(t22,variabili):=block(
          [ss,tn,nd,g22,ch222,ch122,
          g2200,r2222,a,b,c,d,j],
          if not(tensorp(t22)) then
             return("Primo argomento non tensore"),
          nd:dimensione(t22),
          g22:ratsimp((t22[1]+transpose(t22[1]))/2),
          g220:r3(der2(g22,variabili)),
          ss:r3(g220),
          tn:g220+ss,
          ss:r3(ss),
          ch222:ratsimp((tn-ss)/2),
          ch122:transpose(invert(g22).ch222)[1],
          g2200:r4(der3(g220,variabili)),
          r2222:genmatrix(lambda([i,j],ident(nd)),nd,nd),
          for a:1 thru nd do
          for b:1 thru nd do
          for c:1 thru nd do
          for d:1 thru nd do (
             ss:(g2200[b,c][a,d] - g2200[a,c][b,d] +
                 g2200[a,d][b,c] - g2200[b,d][a,c] )/2,
             for j:1 thru nd do ss:ss+
                ch122[j][a,d]*ch222[j][b,c] -
                ch122[j][a,c]*ch222[j][b,d],
             r2222[a,b][c,d]:ss),
          [ ratsimp(r2222),[2,2,2,2,"(r2222)",nd]]
          )$

(%i89) libmia:cons("riemann_2222(t22,variabili)",libmia)$
```

3.6 fa_r22 : genera il tensore di Ricci totalmente covariante.

Definisco il tensore di Ricci doppiamente covariante
seguendo ancora la definizione dell' HEL.

```
(%i90) fa_r22(t22,listavariabili):=block([],
          if controllo then print("Inizia fa_r22"),
          traccia(fa_r1222(t22,listavariabili),1,4)
          )$
```

```
(%i91) libmia:cons("fa_r22(t22,listavariabili)",libmia)$
```

Utilizza la libreria alternativa non ricorsiva.
Dal tensore metrico covariante mi fornisce subito
il tensore di Ricci covariante.

```
(%i92) ricci_22(t22,variabili):=block(
         [nd,r1222,ric22,a,b,sp,j],
          if not(tensorp(t22)) then
             return("Primo argomento non tensore"),
          nd:dimensione(t22),
          r1222:riemann_1222(t22,variabili)[1],
          r22:zeromatrix(nd,nd),
          for a:1 thru nd do
          for b:a thru nd do (
              sp:0,
              for j:1 thru nd do sp:sp+r1222[j,a][b,j],
              r22[a,b]:sp,
              r22[b,a]:sp
              ),
          [ ratsimp(r22),[2,2,"(Ricci 22)",nd ] ]
          )$
```

3.7 fametrica: costruttore di metrica. Genera una lista che include tutto quello che serve per fare derivate covarianti.

Il calcolo differenziale tensoriale si basa su una
funzione, fametrica, che genera la metrica usata per
calcolare la derivata covariante.
Questa funzione deve ricevere in argomento una matrice
che, per sicurezza, viene simmetrizzata e usata come
tensore metrico covariante della metrica.

```
(%i93) fa_metrica(mg,listavariabili):=block(
          [metrica,m2,m3,m4,m4,m6,m7,m8,m9,ma],
          if not(matrixp(mg)) then
             return("Errore, primo arg. non matrice"),
          if not(listp(listavariabili)) then
             return("Errore, secondo arg. non lista variabili"),
          if length(listavariabili)#length(mg) then
             return("Errore: incongruenza tra matrice e lista variabili"),
          if controllo then print("Inizia g22"),
          m2: [ratsimp((mg + transpose(mg) )/2),
              [2,2,"(tensore metrico covariante)",length(mg)]],
          m2: trigsimp(m2),
          if controllo then print("Inizia g11"),
          m3: [ratsimp(invert(m2[1])),
              [1,1,"(tensore metrico controvariante)",length(mg)]],
          if controllo then print("Inizia g220"),
          m4: fa_diff(m2,listavariabili,"g220"),
          if controllo then print("Inizia g2200"),
          m5: fa_diff(m4,listavariabili,"g2200"),
          m6: fa_ch222(m4),
          commentalo(m6,"(Christoffel prima specie)"),
          if controllo then print("Inizia ch122"),
          m7: scala(m3,2,m6,1),
          commentalo(m7,"(Christoffel seconda specie)"),
          m8: fa_r1222(m2,listavariabili),
          m9: fa_r2222(m2,listavariabili),
          if controllo then print("Inizia r22"),
          ma: traccia(m8,1,4),
          commentalo(ma,"(Ricci totalmente covariante)"),
          mb:[ratsimp(sqrt(abs(determinant(m2[1])))),
             ["sqrt(|determinant(g22)|)",length(mg)]],
          metrica:[listavariabili,m2,m3,m4,m5,m6,m7,m8,m9,ma,mb],
          infometrica(),
          metrica)$

(%i94) infometrica():=block([],
          print("Ha creato un vettore di undici elementi:"),
          print("[1] : lista delle variabili"),
          print("[2] : tensore metrico covariante"),
          print("[3] : tensore metrico controvariante"),
          print("[4] : derivate prime del tens. metrico cov."),
          print("[5] : derivate seconde del tens. m. cov."),
          print("[6] : simboli di Christoffel di prima specie"),
          print("[7] : simboli di Christoffel di seconda specie"),
          print("[8] : Riemann col solo primo indice controvariante"),
          print("[9] : Riemann totalmente controvariante"),
          print("[10]: Ricci totalmente covariante"),
          print("[11]: sqrt(abs(determinate del tens.metr.cov.))")
          )$

(%i95) libmia:cons("fa_metrica(mg,listavariabili)",libmia)$
```

3.8 Derivata covariante

Per convenzione la cifra che segnala un indice ottenuto da derivazione covariante è 8 e quello trasformato in controvariante è 9. Dunque xx28 è un vettore covariante (xx2) diventato tensore del secondo ordine per derivazione covariante ed analogamente xx18 è un vettore controvariante (xx1), derivato covariantemente. Se alzo l'ultimo indice di xx28 ottengo xx29, etc...

Dato che la derivazione covariante produce tensori,
è possibile derivare qualsiasi tensore usando i simboli
di Christoffel in due modi diversi a seconda della
natura covariante o controvariante di un dato
indice.
Per fare la derivazione occorrono tanti simboli di Christoffel
quanti sono gli indici del tensore e dunque per derivare
uno scalare non occorrono simboli di Christoffel ossia
la derivazione di uno scalare coincide con quella ordinaria.
Esaminiamo i due casi distinti usando le convenzioni della
presente libreria che impone che i tensori di qualunque
ordine siano delle liste di qualcosa, scalari o matrici
e che l'ultimo elemento della lista sia a sua volta una
lista che contiene informazioni sulla natura degli
indici del tensore.

Dunque:

A[a] è uno scalare o un vettore del primo ordine. Se è
uno scalare esiste solo A[1] mentre se è un vettore
esistono anche le componenti A[2]...A[nd] dove nd
è il numero di dimensioni dello spazio, ovvero 3
in meccanica classica e 4 in meccanica relativistica.
B[a][b,c] è un tensore del secondo o del terzo ordine.
C[a][b,c][d,e] è un tensore del quarto o del quinto ordine.

Per accedere alla lista dei tipi degli indici
bisogna prendere l'ultimo elemento del primo indice ossia
A[length(A)] o B[length(B)] o C[length[C)] è la
lista dei tipi degli indici, qualunque sia l'ordine
del tensore A o B o C.
L'ultima delle componenti, se il tensore ha ordine pari
è sempre 2 mentre se ha ordine dispari è sempre
data da nd+1.
In pratica però è meglio non fare affidamento su
questa regola e prendere sempre l'ultimo indice
della lista perché in questo modo è possibile
usare anche liste più lunghe del minimo indispensabile
e memorizzare negli elementi sovrabbondanti i
dati che si desidera associare in più al
tensore.
Notare che in Maxima è ammesso omettere gli indici
tra parentesi quadra per cui se scrivo B[1] ottengo
l'unica o la prima matrice del tensore B (o un singolo
dato numerico se B è uno scalare o un vettore); la
matrice (o scalare) è l'unica se B è un tensore di
ordine pari ossia ad esempio uno scalare o un tensore
del secondo ordine ed è la prima se B è un tensore di
ordine dispari ossia, per esempio un vettore o un
tensore del terzo ordine.

Per la derivata di un vettore covariante A2, ossia di un tensore di ordine 1 dove la cifra 2 ricorda che il suo primo ed unico indice è covariante, la formula è questa:

```
A2a8b = A2a0b - ch1j2a2b*A2j
```

ovvero in Maximese (essendo 8 la cifra che indica derivata covariante) :

```
A28[1][a,b] = A20[1][a,b] - ch122[j][a,b]*A2[j]
```

dove la cifra 0 specifica la derivata ordinaria e ch122 rappresenta il simbolo di Christoffel di seconda specie avente la cifra 1 che ricorda che il suo primo indice è (pseudo)controvariante e la cifra 2 che ricorda che il secondo e terzo indice sono (pseudo)covarianti.

Per la derivata di un tensore del secondo ordine totalmente covariante A22, la formula è questa:

```
A2a2b8c = A2a2b0c - ch1j2a2c*A2j2b -
                    ch1j2b2c*A2a2j
```

ovvero in Maximese, dato il tensore A[1][a,b] ottengo:

```
A228[a][b,c] = A220[a][b,c] -
                   ch122[j][a,c]*A22[1][j,b] -
                   ch122[j][b,c]*A22[1][a,j]
```

per cui è facile intuire la regola generale da applicarsi ad un tensore di ordine qualsiasi.

La derivata di un tensore del terzo ordine totalmente covariante ossia A222, la formula è questa:

```
A2a2b2c2d = A2a2b2c0d - ch1j2a2d*A2j2b2c -
                        ch1j2b2d*A2a2j2c -
                        ch1j2c2d*A2a2b2j
```

ovvero in Maximese, dato il tensore A[a][b,c]:

```
A2228[1][a,b][c,d] = A2220[1][a,b][c,d] -
                     ch122[j][a,d]*A222[j][b,c] -
                     ch122[j][b,d]*A222[a][j,c] -
                     ch122[j][c,d]*A222[a][b,j]
```

A questo punto è facile generalizzare il metodo a tensori di qualsiasi ordine.

La derivata di un vettore controvariante si fa invece
in questo modo (usando uno dei due indici
pseudocovarianti del simbolo di Christoffel che è
uno pseudo tensore) :

```
   A1a8b = A1a0b + ch1a2j2b*A1j
```

Notare che il simbolo di Christoffel è simmetrico nei
suoi due ultimi indici trattati come covarianti, per cui
avrei potuto scrivere anche in questo modo (è solo una
questione estetica)

```
   A1a8b = A1a0b + ch1a2b2j*A1j
```

ossia in Maximese, detto A1[a] il vettore da derivare
ho la seguente formula

```
  A18[1][a,b] = A10[1][a,b] + ch122[a][j,b]*A1[j]
```

La derivata di un tensore del secondo ordine totalmente
controvariante A1a1b si fa, di conseguenza in questo modo:

```
   A1a1b8c = A1a1b0c + ch1a2j2c*A1j1b
                     + ch1b2j2c*A1a1j
```

ossia in Maximese, dato A11[1][a,b] si ha:

```
   A118[a][b,c] = A110[a][b,c] +
                      ch122[a][j,c]*A11[1][j,b] +
                      ch122[b][j,c]*A11[1][a,j]
```

Attenzione: il primo contributo è dato da A11[1][j,b]
che non è equivalente a A11[1][b,j] a meno che il
tensore del secondo ordine non sia, in pratica,
una matrice simmetrica.

```
(%i96) diffcov(tn,metrica,nome):=block([indi,o,ch122,tr,lp,j,temp],
           if not(tensorp(tn))
              then return("Errore: non tensore"),
           indi:indici(tn),
           o:ordine(tn),
           if dimensione(tn)#dimensione(metrica[2]) then
              return("Errore, dimensioni incongruenti"),
           ch122:metrica[7],
           tr:rest(fa_diff(tn,metrica[1],"-"),-1),
           lp:makelist(mod(j,o+1)+1,j,1,o+1),
           lp[1]:1,lp[o+1]:2,
           for j:1 thru o do (
               if mod(indi[j],2)=0 then (
                   tr:tr-rest(permuta(scala(ch122,1,tn,j),lp,o),-1)
                   )
               else (
                   tr:tr+rest(permuta(scala(ch122,2,tn,j),lp,o),-1)
                   ),
               temp:lp[j],lp[j]:lp[j+1],lp[j+1]:temp
               ),
           endcons(append(indi,
              [8,nome,dimensione(tn)]),ratsimp(tr))
           )$
```

```
(%i97) libmia:cons("diffcov(tn,metrica,nome)",libmia)$
```

Per mettere alla dura prova questa libreria proviamo a
verificare alcune identità che debbono sempre sussistere.

A pag. 344, formula (92,4), il Landau afferma.
La somma ciclica di qualsiasi terna di indici del tensore
di Riemann r2222 deve essere sempre nulla.
In altre parole :

```
    r2222[1][a,b][c,d] +
    r2222[1][a,d][b,c] + r2222[1][a,c][d,b] = 0
```

Supponiamo di aver calcolato r2222....

Usando la funzione permuta si ottiene in generale un tensore
del quarto ordine che, a quanto dice il Landau ma non solo
lui, dovrebbe risultare totalmente nullo.
Con le funzioni di questa libreria si dovrebbe fare così:

```
rest(r2222,-1)+
rest(permuta(r2222,[1,3,4,2],""),-1)+
rest(permuta(r2222,[1,4,2,3],""),-1);
```

Per provare la "identità di Bianchi" ossia la seguente relazione:

```
    r12228[a][b,c][d,e] +
    r12228[a][b,e][c,d] +
    r12228[a][b,d][e,c] = 0
```

bisogna fare la derivata covariante del tensore di Riemann col solo primo indice controvariante ossia prelevarlo dalla metrica.
Con questa libreria si fa così:

```
r12228:ratsimp(diffcov(r1222,metrica,"r12228"))$
```

Notare che è un calcolo lungo perché si tratta di calcolare un tensore del quinto ordine ossia con 4^5 = 1024 componenti!

Noto r12228, ecco come va calcolata l'identità di Bianchi con la libreria ossia permutando gli indici. In quattro dimensioni questo tensore del quinto ordine ha 1024 componenti che dovrebbero risultare tutte nulle!

```
idbianchi:rest(r12228,-1)+
          rest(permuta(r12228,[1,2,4,5,3],""),-1)+
          rest(permuta(r12228,[1,2,5,3,4],""),-1)$
```

Stupefacente! Vengono tutti zeri come deve essere...

Mi sembra che questo risultato sia una buona dimostrazione della correttezza della libreria...

Ora uso l'invariante del tensore di Ricci ossia r1i2i
che ho chiamato rinva.Supponiamo di aver già calcolato
il tensore di Ricci totalmente covariante ossia r22.
Allora, supponendo che g11 sia il tensore metrico
controvariante:

```
r12: scala(g11,2,r22,1)$
rinva : traccia(scala(g11,2,r22,1),1,2)$
```

Derivando covariantemente rinva, usando la metrica
appropriata, si ottiene un vettore ossia:

```
rinva8: ratsimp(diffcov(rinva,metrica,"rinva8"))$
```

Notare che si sarebbe ottenuto lo stesso vettore
usando la derivata ordinaria ossia usando la
fa_diff(tn,listavariabili,comme) nel seguente modo:

```
rinva0 : fa_diff(rinva,listavariabili,"rinva0")
```

infatti in caso di scalari la derivata covariante
coincide con la derivata ordinaria.

La derivata covariante del tensore di Ricci in forma
mista è data da r128 ossia

```
r128:ratsimp(diffcov(scala(g11,2,r22,1),
             metrica,"r128"))$
```

L'identità che dobbiamo verificare è la seguente:

```
2*r128[a][b,a]-rinva8[b] = 0
```

Con questa libreria si fa così:

```
ratsimp(2*rest(traccia(r128,1,3),-1)-rest(rinva8,-1));
```

Notare che con la rest(list,-1) si taglia via dalla
lista il suo ultimo elemento ossia quello che contiene
le info sul tipo di indici del tensore.
Quello che resta è dunque qualcosa di trattabile
algebricamente.

Se quindi viene un vettore di zeri è un trionfo...

E, fatte le verifiche, viene proprio un vettore
di zeri e dunque, salvo bachi nascosti, tutto,
fino qui... FUNZIONA!

3.9 Divergenza di un tensore

Crea la funzione divergenza di un tensore rispetto ad un
dato indice, nota la metrica e dunque la dimensione,
l'ordine del tensore e il tensore metrico controvariante.
Fa uso della derivata covariante per cui questa funzione
è applicabile a qualunque tensore di ordine positivo.

```
(%i98) divergenza(tn,ki,metrica):=block([k,tm,j],
           if not(tensorp(tn)) then
              return("Errore: non tensore"),
           if not(numberp(ki)) then
              return("Errore: non numero intero"),
           k:floor(ki),
           if 1>k then
              return("Errore: indice non positivo"),
           j:indici(tn),
           if k>length(j) then
              return("Errore: indice troppo elevato"),
           if mod(j[k],2)=1 then tm:diffcov(tn,metrica,"(div)")
           else ( tm:scala(metrica[3],1,tn,k),
                  tm:diffcov(tm,metrica,"(div)")),
           traccia(tm,k,ordine(tm))
           )$
```

```
(%i99) libmia:cons("divergenza(tn,ki,metrica)",libmia)$
```

In complesso le funzioni attualmente definite in questa
libreria sono queste:

```
(%i100) libmia:sort(libmia);
(%o100) [ada(la,per),commentalo(tn,comme),commento(tn),
diffcov(tn,metrica,nome),dimensione(tn),divergenza(tn,ki,metrica),
fa_ch122(g22,listavariabili),fa_ch222(g220),fa_diff(tn,variabili,comme)
,fa_metrica(mg,listavariabili),fa_r1222(t22,listavariabili),
fa_r22(t22,listavariabili),fa_r2222(t22,listavariabili),indici(tn),
listadatensore(tn),ordine(tn),permuta(tn,per,com),
riemann_1222(t22,variabili),riemann_2222(t22,variabili),
scala(ta,ia,tb,ib),tassegna(tn,val,p),tcontrov(lv,metrica),
tcov(lv,metrica),tensoredalista(lista,tipi,comme,ndim),tensorp(tn),
tmat11(mat,metrica),tmat12(mat,metrica),tmat21(mat,metrica),
tmat22(mat,metrica),tmeno(tena,tenb),tprod(tscala,tenb),
traccia(tn,ih,ik),tsca(sc,metrica),tsomma(tena,tenb),tvale(tn,p),
zerotensor(tipi,comme,ndim)]
```

3.10 Conclusione...provvisoria

Chiunque ami le formule matematiche ricche di simmetrie non può
non ammirare la bellezza della matrice del tensore di Ricci
quando la metrica è quella di Reissner Nordstrom ma non nella
forma originale bensì nella forma proposta da Schild.

La matrice del tensore metrico, nel caso della variante di Schild, non è più diagonale bensì piena e questo ha un effetto piuttosto... devastante sulla velocità del calcolo effettuato da Maxima.... ma il risultato è il poter fare calcoli in coordinate "quasi cartesiane" visto che al tendere all'infinito della distanza dal buco nero la metrica ritorna ad essere quella pseudoeuclidea della Relatività Speciale

Anche se i normali buchi neri macroscopici (quello al centro della via Lattea, ad esempio) sono neutri ritengo molto opportuno ipotizzare che il buco nero gigantesco abbia la carica di almeno un elettrone... per i seguenti motivi.

In Relatività lo stesso problema fisico puo' essere trattato con infinite metriche diverse... come nella meccanica classica. Infatti, anche in meccanica classica si possono usare i piu' disparati sistemi di riferimento che producono, derivando lo Jacobiano, diversissimi tensori metrici. Cito a memoria... sistema cilindrico, sferico, parabolico, toroidale, ellissoidale etc... Di tutti questi sistemi e di molti altri, con il package ctensor di Maxima è possibile calcolare l'appropriato tensore metrico.

Quello pero' che NON CAMBIA è il valore del prodotto scalare ossia il modulo delle forze in azione. In Relatività, giustamente, più che di "scalare" si parla di "invariante" perché appunto, una quantità invariante non cambia qualunque sia il sistema di riferimento e dunque la sua metrica associata.

Ad una data distanza dal buco nero esiste una forza necessaria per impedire che l'oggetto cada nel buco nero. Se anche il buco nero ha la carica di un solo elettrone, dato che la forza elettrostatica repulsiva dipende dal prodotto delle due cariche (dello stesso segno) dei due oggetti che si respingono. Dunque esisterà sempre una carica della particella tale da impedire che la particella acceleri verso il buco nero ossia cada in esso.

Ma l'entità di questa carica è UN INVARIANTE ! ed è un invariante il modulo della forza necessaria per impedire alla particella ferma di cominciare ad accelere

Numericamente il problema ossia il calcolo
di questa forza è facile perchè si tratta di
risolvere un problemino di ELETTROSTATICA,
sia pure in uno spazio deformato dalla presenza
del grande (o piccolo) buco nero.

Siccome la forza è un vettore il suo modulo
è un invariante e dunque non cambia qualunque
sia il sistema di riferimento (da cui deriva la
metrica) che sto usando.

Insomma ..non importa che stia usando, in pratica
la metrica di Schwarzschild o quella di Reissner
Nordstrom o quella di chiunque altro inventi
una sua metrica... Si pensi a quella del
reverendo Lemaitre:
http://en.wikipedia.org/wiki/Lemaitre_metric

IL MODULO DELLA FORZA CHE EQUILIBRA L'ACCELERAZIONE
DI GRAVITA' NON CAMBIA ossia LA RELATIVITA' NON E'
ASSOLUTAMENTE RELATIVA ma INVARIANTE ed ASSOLUTA !

3.11 Chiusura file di salvataggio.

```
(%i101) print("Fine della libreria tensoriale. Vedere libmia")$
Fine della libreria tensoriale. Vedere libmia
```

```
(%i102) closefile();
Finished dribbling to c:/xmaxima/libtensori.mc.
NIL
(%o102) done
```

Non resta che utilizzare questa libreria effettuando
un load(salvoqui).