MapLevelParts

■ `MapLevelParts[` *function,* {*topposition, levelpositions*}`][` *expr* `]` will map the function onto the selected level positions in an expression.

Levelpositions is an integer list of the selected parts. The function is applied to them as a group and they are replaced with a single new expression. Other parts not specified on the list are left unchanged.

MapLevelParts is useful when you wish to apply SimplifyTensorSum to just a selected set of terms in a sum. It is easier and more controlled to simplify such sums when you are not dealing with disparate terms. It is also useful in applying MetricSimplify or KroneckerAbsorb to a selected set of factors in a term.

If you use TraditionalForm output the display order of terms is not always the same as the internal order. Add `// StandardForm` to the output you will be operating on to get the internal order.

See also: `SimplifyTensorSum`, `TensorSimplify`, `UpDownSwap`, `IndexChange`, `SymmetrizeSlots`, `MetricSimplify`, `MapLevelPatterns`.

Examples

```
In[1]:= Needs["TensorCalculus4`Tensorial`"]

In[2]:= DefineTensorShortcuts[{{λ, κ}, 1}, {g, 2}]
```

The following is an example of using `MapLevelParts`.

For smooth functions the order of partial differentiation does not matter. It does matter for covariant differentiation. In general order independence is not true for covariant differentiation. For example, for a contravariant vector field...

```
In[3]:= Print["Covariant differentiation is not commutative"]
        CovariantD[λu[a], {b, c}] ≠ CovariantD[λu[a], {c, b}]
        Print["Expand, rearrange and use symmetry on Γ"]
        ExpandCovariantD[{x, δ, g, Γ}, {d, e}] /@ %%;
        # - Part[%, 2] & /@ % // ExpandAll;
        % // SymmetrizeSlots[Γ, 3, Symmetric[2, 3]]
        Print["Factor the first four terms, and use SimplifyTensorSum
           on the other terms\nFirst we factored the first four terms."]
        %% // MapLevelParts[Factor, {1, {1, 2, 3, 4}}]
        Print[
         "Then we used SimplifyTensorSum on the last four terms, which reduced to 0."]
        %% // MapLevelParts[SimplifyTensorSum, {1, {2, 3, 4, 5}}]
```

```
Covariant differentiation is not commutative
```

$$\text{Out[4]=}\quad \lambda^{a}{}_{;bc} \neq \lambda^{a}{}_{;cb}$$

```
Expand, rearrange and use symmetry on Γ
```

$$\text{Out[8]=}\quad \Gamma^{a}{}_{ce}\,\Gamma^{e}{}_{bd}\,\lambda^{d} - \Gamma^{a}{}_{be}\,\Gamma^{e}{}_{cd}\,\lambda^{d} + \lambda^{d}\,\frac{\partial \Gamma^{a}{}_{bd}}{\partial x^{c}} - \lambda^{d}\,\frac{\partial \Gamma^{a}{}_{cd}}{\partial x^{b}} - \frac{\partial^{2} \lambda^{a}}{\partial x^{b}\,\partial x^{c}} + \frac{\partial^{2} \lambda^{a}}{\partial x^{c}\,\partial x^{b}} - \Gamma^{a}{}_{cd}\,\frac{\partial \lambda^{d}}{\partial x^{b}} + \Gamma^{a}{}_{bd}\,\frac{\partial \lambda^{d}}{\partial x^{c}} + \Gamma^{a}{}_{ce}\,\frac{\partial \lambda^{e}}{\partial x^{b}} - \Gamma^{a}{}_{be}\,\frac{\partial \lambda^{e}}{\partial x^{c}} \neq 0$$

```
Factor the first four terms, and use SimplifyTensorSum on the other terms
First we factored the first four terms.
```

$$\text{Out[10]=}\quad \lambda^{d}\left(\Gamma^{a}{}_{ce}\,\Gamma^{e}{}_{bd} - \Gamma^{a}{}_{be}\,\Gamma^{e}{}_{cd} + \frac{\partial \Gamma^{a}{}_{bd}}{\partial x^{c}} - \frac{\partial \Gamma^{a}{}_{cd}}{\partial x^{b}}\right) - \frac{\partial^{2} \lambda^{a}}{\partial x^{b}\,\partial x^{c}} + \frac{\partial^{2} \lambda^{a}}{\partial x^{c}\,\partial x^{b}} - \Gamma^{a}{}_{cd}\,\frac{\partial \lambda^{d}}{\partial x^{b}} + \Gamma^{a}{}_{bd}\,\frac{\partial \lambda^{d}}{\partial x^{c}} + \Gamma^{a}{}_{ce}\,\frac{\partial \lambda^{e}}{\partial x^{b}} - \Gamma^{a}{}_{be}\,\frac{\partial \lambda^{e}}{\partial x^{c}} \neq 0$$

```
Then we used SimplifyTensorSum on the last four terms, which reduced to 0.
```

$$\text{Out[12]=}\quad \lambda^{d}\left(\Gamma^{a}{}_{ce}\,\Gamma^{e}{}_{bd} - \Gamma^{a}{}_{be}\,\Gamma^{e}{}_{cd} + \frac{\partial \Gamma^{a}{}_{bd}}{\partial x^{c}} - \frac{\partial \Gamma^{a}{}_{cd}}{\partial x^{b}}\right) - \frac{\partial^{2} \lambda^{a}}{\partial x^{b}\,\partial x^{c}} + \frac{\partial^{2} \lambda^{a}}{\partial x^{c}\,\partial x^{b}} - \Gamma^{a}{}_{cd}\,\frac{\partial \lambda^{d}}{\partial x^{b}} + \Gamma^{a}{}_{bd}\,\frac{\partial \lambda^{d}}{\partial x^{c}} + \Gamma^{a}{}_{ce}\,\frac{\partial \lambda^{e}}{\partial x^{b}} - \Gamma^{a}{}_{be}\,\frac{\partial \lambda^{e}}{\partial x^{c}} \neq 0$$

In general the term in brackets will not be zero. (TensorSimplify actually automatically selects groups of terms on which to apply SimplifyTensorSum.)

Sometimes when using `MetricSimplify`, or `KroneckerAbsorb`, it is ambiguous as to which factors are to be operated on. In the following the index on the κ vector is lowered.

```
In[13]:= gdd[a, b] κu[a] λu[b]
         % // MetricSimplify[g]
```

$$\text{Out[13]=}\quad g_{ab}\,\kappa^{a}\,\lambda^{b}$$

$$\text{Out[14]=}\quad \kappa_{b}\,\lambda^{b}$$

If we wish to lower the λ index instead, we could use...

```
In[15]:= gdd[a, b] κu[a] λu[b]
         % // MapLevelParts[MetricSimplify[g], {{1, 3}}]
```

$$\text{Out[15]=}\quad g_{ab}\,\kappa^{a}\,\lambda^{b}$$

$$\text{Out[16]=}\quad \kappa^{a}\,\lambda_{a}$$

```
In[17]:= ClearTensorShortcuts[{{λ, κ}, 1}, {g, 2}]
```

MapLevelPatterns

- `MapLevelPatterns[`*function,* {*topposition,* {*pattern*}}`][`*expr*`]` will map the function onto the selected level positions that match the pattern in an expression.

The pattern is used to determine a set of levelpositions at topposition. The function is applied to them as a group and they are replaced with a single new expression. Other level parts are left unchanged.

MapLevelPatterns will often be a more convenient form of MapLevelParts.

MapLevelPatterns is useful when you wish to apply SimplifyTensorSum to just a selected set of terms in a sum. It is easier and more controlled to simplify such sums when you are not dealing with disparate terms. It is also useful in applying MetricSimplify or KroneckerAbsorb to a selected set of factors in a term.

See also: `SimplifyTensorSum`, `TensorSimplify`, `UpDownSwap`, `IndexChange`, `SymmetrizeSlots`, `MetricSimplify`, `MapLevelParts`.

Examples

```
In[1]:= Needs["TensorCalculus4`Tensorial`"]

In[2]:= DefineTensorShortcuts[{{λ, ϰ, x}, 1}, {g, 2}]
```

The following is an example of using `MapLevelPatterns`. This is the same example as in `MapLevelParts`.

For smooth functions the order of partial differentiation does not matter. It does matter for covariant differentiation. In general order independence is not true for covariant differentiation. For example, for a contravariant vector field...

```
In[3]:= Print["Covariant differentiation is not commutative"]
        CovariantD[λu[a], {b, c}] ≠ CovariantD[λu[a], {c, b}]
        Print["Expand, rearrange and use symmetry on Γ"]
        ExpandCovariantD[{x, δ, g, Γ}, {d, e}] /@ %%;
        # - Part[%, 2] & /@ % // ExpandAll;
        % // SymmetrizeSlots[Γ, 3, Symmetric[2, 3]]
        Print[
         "Use MapLevelPatterns to simplify the terms containing λ derivatives and factor"]
        %% // MapLevelPatterns[SimplifyTensorSum, {1, {a_. PartialD[_][λu[_], xu[_]]}}]
        MapAt[Factor, %, 1]
```

Covariant differentiation is not commutative

$$\text{Out[4]=}\quad \lambda^{a}{}_{;bc} \neq \lambda^{a}{}_{;cb}$$

Expand, rearrange and use symmetry on Γ

$$\text{Out[8]=}\quad \Gamma^{a}{}_{ce}\,\Gamma^{e}{}_{bd}\,\lambda^{d} - \Gamma^{a}{}_{be}\,\Gamma^{e}{}_{cd}\,\lambda^{d} + \lambda^{d}\,\frac{\partial \Gamma^{a}{}_{bd}}{\partial x^{c}} - \lambda^{d}\,\frac{\partial \Gamma^{a}{}_{cd}}{\partial x^{b}} - \frac{\partial^{2}\lambda^{a}}{\partial x^{b}\,\partial x^{c}} + \frac{\partial^{2}\lambda^{a}}{\partial x^{c}\,\partial x^{b}} - \Gamma^{a}{}_{cd}\,\frac{\partial \lambda^{d}}{\partial x^{b}} + \Gamma^{a}{}_{bd}\,\frac{\partial \lambda^{d}}{\partial x^{c}} + \Gamma^{a}{}_{ce}\,\frac{\partial \lambda^{e}}{\partial x^{b}} - \Gamma^{a}{}_{be}\,\frac{\partial \lambda^{e}}{\partial x^{c}} \neq 0$$

Use MapLevelPatterns to simplify the terms containing λ derivatives and factor

$$\text{Out[10]=}\quad \Gamma^{a}{}_{ce}\,\Gamma^{e}{}_{bd}\,\lambda^{d} - \Gamma^{a}{}_{be}\,\Gamma^{e}{}_{cd}\,\lambda^{d} + \lambda^{d}\,\frac{\partial \Gamma^{a}{}_{bd}}{\partial x^{c}} - \lambda^{d}\,\frac{\partial \Gamma^{a}{}_{cd}}{\partial x^{b}} - \frac{\partial^{2}\lambda^{a}}{\partial x^{b}\,\partial x^{c}} + \frac{\partial^{2}\lambda^{a}}{\partial x^{c}\,\partial x^{b}} \neq 0$$

$$\text{Out[11]=}\quad \Gamma^{a}{}_{ce}\,\Gamma^{e}{}_{bd}\,\lambda^{d} - \Gamma^{a}{}_{be}\,\Gamma^{e}{}_{cd}\,\lambda^{d} + \lambda^{d}\,\frac{\partial \Gamma^{a}{}_{bd}}{\partial x^{c}} - \lambda^{d}\,\frac{\partial \Gamma^{a}{}_{cd}}{\partial x^{b}} - \frac{\partial^{2}\lambda^{a}}{\partial x^{b}\,\partial x^{c}} + \frac{\partial^{2}\lambda^{a}}{\partial x^{c}\,\partial x^{b}} \neq 0$$

In general the term in brackets will not be zero.

Sometimes when using `MetricSimplify`, or `KroneckerAbsorb`, it is ambiguous as to which factors are to be operated on. In the following the index on the κ vector is lowered.

```
In[12]:= gdd[a, b] κu[a] λu[b]
         % // MetricSimplify[g]
```

$$\text{Out[12]=}\quad g_{ab}\,\kappa^{a}\,\lambda^{b}$$

$$\text{Out[13]=}\quad \kappa_{b}\,\lambda^{b}$$

If we wish to lower the λ index instead, we could pick out the relevant factors...

```
In[14]:= gdd[a, b] κu[a] λu[b]
         % // MapLevelPatterns[MetricSimplify[g], {{ gdd[_, _] | λu[_]}}]
```

$$\text{Out[14]=}\quad g_{ab}\,\kappa^{a}\,\lambda^{b}$$

$$\text{Out[15]=}\quad \kappa^{a}\,\lambda_{a}$$

If there are no matches in the topposition a warning message is issued and the expression is returned unchanged. In this case topposition referred to the wrong term. (TraditionalForm changes the display order of terms.) In other cases the pattern may have been specified incorrectly.

```
In[16]:= 3 + gdd[a, b] κu[a] λu[b] // TraditionalForm
         % // MapLevelPatterns[MetricSimplify[g], {1, { gdd[_, _] | λu[_]}}]
```

Out[16]//TraditionalForm=

$$g_{a\,b}\,\kappa^{a}\,\lambda^{b}+3$$

```
MapLevelPatterns::nomatch : There were no matches for g__ | λ- in 3
```

```
Out[17]= 3 + g_{a b} κ^a λ^b
```

MetricSimplify

- `MetricSimplify[`*g, dopartials : False* `][`*expression*`]` will raise or lower indices when products with the metric tensor g appear.

g is the label of the metric tensor. The routine assumes that any second order tensor with the label g is a metric tensor. One could also use other labels to stand for the metric tensor, for example η or h.

Simplification occurs only if the two indices of `g` have the same flavor.

Up/down and down/up metric tensors will act as Kroneckers, performing index substitution. Up/down and down/up metric tensors that contain base indices will simplify to 0 or 1. Hence you can use MetricSimplify to evaluate Kroneckers.

The decision to use `MetricSimplify` may depend upon the particular tensor forms that have stored values. You may often wish to use `MetricSimplify` on only portions of an expression using `MapLevelParts`.

If both indices of a metric tensor are dummy indices in an expression, then the lowest sort order index is used as the active dummy index in the simplification.

MetricSimplify will not commute with `PartialD` or `TotalD` unless the optional argument dopartials is set to true. It will commute with CovariantD and AbsoluteD since the covariant derivative of the metric tensor is zero.

See also: `SetTensorValues`, `SetTensorValueRules`, `EvaluateDotProducts`, `MapLevelParts`.

Examples

```
In[1]:= Needs["TensorCalculus4`Tensorial`"]
```

Save the settings and declare base indices and flavors...

```
In[2]:= oldindices = CompleteBaseIndices;
        oldflavors = IndexFlavors;
        ClearIndexFlavor /@ oldflavors;
        DeclareBaseIndices[{1, 2, 3}]
        DeclareIndexFlavor /@ {{red, Red}, {rocket, SuperStar}};

In[7]:= DefineTensorShortcuts[{{A, S, x, y}, 1}, {{g, h, η, δ, S}, 2}, {S, 4}]
        labs = {x, δ, g, Γ};
```

Lowering indices...

```
In[9]:= gdd[i, j] xu[i]
        % // MetricSimplify[g]

Out[9]= g_{i j} x^i

Out[10]= x_j
```

Raising indices....

```
In[11]:= guu[i, j] xd[j]
         % // MetricSimplify[g]

Out[11]= g^{i j} x_j

Out[12]= x^i

In[13]:= guu[i, j] xd[j] Su[k] // ToFlavor[red]
         % // MetricSimplify[g]

Out[13]= g^{i j} S^k x_j

Out[14]= S^k x^i

In[15]:= guu[i, j] guu[k, l] Sdddd[i, j, k, l]
         % // MetricSimplify[g]

Out[15]= g^{i j} g^{k l} S_{i j k l}

Out[16]= S^j_j^l_l
```

Mixed up/down metric tensors act like Kroneckers.

```
In[17]:= {gud[i, j] xu[j], gdu[i, j] xu[i], gud[i, j] xd[i], gdu[i, j] xd[j]}
         % // MetricSimplify[g]

Out[17]= {g^i_j x^j, g_i^j x^i, g^i_j x_i, g_i^j x_j}

Out[18]= {x^i, x^j, x_j, x_i}
```

MetricSimplify is mapped over equations, lists and sums. Each side of an equation is expanded.

```
In[19]:= gud[i, j] (xu[j] + yu[j]) == xu[i] + yu[i]
         % // MetricSimplify[g]

Out[19]= g^i_j (x^j + y^j) == x^i + y^i

Out[20]= True
```

When an up/down or down/up metric tensor contains base indices it is evaluated to 0 or 1.

```
In[21]:= {gud[1, 1], gud[1, 2], guu[1, 1], gdd[1, 2], gud[red@1, red@1], gdu[red@1, red@2]};
         Thread[% → (% // MetricSimplify[g])]

Out[22]= {g^1_1 → 1, g^1_2 → 0, g^{1 1} → g^{1 1}, g_{1 2} → g_{1 2}, g^1_1 → 1, g_1^2 → 0}
```

So, MetricSimplify is an easy method to evaluate Kroneckers.

```
In[23]:= δud[3, 3]
         % // MetricSimplify[δ]

Out[23]= δ^3_3

Out[24]= 1
```

The base indicies don't have to be integers.

```
In[25]:= DeclareBaseIndices[{ρ, θ, ϕ}]
         {gud[ρ, ρ], gud[ρ, θ], guu[ρ, ρ], gdd[ρ, ϕ], gud[red@ρ, red@ρ], gdu[red@θ, red@ϕ]};
         Thread[% → (% // MetricSimplify[g])]
         DeclareBaseIndices[{1, 2, 3}]
```

$$\text{Out[27]=}\ \{g^{\rho}{}_{\rho} \to 1,\ g^{\rho}{}_{\theta} \to 0,\ g^{\rho\rho} \to g^{\rho\rho},\ g_{\rho\phi} \to g_{\rho\phi},\ g^{\rho}{}_{\rho} \to 1,\ g_{\theta}{}^{\phi} \to 0\}$$

The index flavors must match...

```
In[29]:= gdd[i, j] (Sdu[m, i] // ToFlavor[red])
         % // MetricSimplify[g]
```

$$\text{Out[29]=}\ g_{i\,j}\,S_{m}{}^{i}$$

$$\text{Out[30]=}\ g_{i\,j}\,S_{m}{}^{i}$$

```
In[31]:= gdd[i, j] Sdu[m, i] // ToFlavor[rocket]
         % // MetricSimplify[g]
```

$$\text{Out[31]=}\ g_{i^*\,j^*}\,S_{m^*}{}^{i^*}$$

$$\text{Out[32]=}\ S_{m^*\,j^*}$$

Both indices on the metric matrix must be the same flavor.

```
In[33]:= gdd[red@i, j] xu[j]
         % // MetricSimplify[g]
```

$$\text{Out[33]=}\ g_{i\,j}\,x^{j}$$

$$\text{Out[34]=}\ g_{i\,j}\,x^{j}$$

MetricSimplify can use base indices as the replacement value.

```
In[35]:= guu[3, i] Sdd[i, j]
         % // MetricSimplify[g]
```

$$\text{Out[35]=}\ g^{3\,i}\,S_{i\,j}$$

$$\text{Out[36]=}\ S^{3}{}_{j}$$

MetricSimplify commutes with Covariant and Absolute derivatives. This can be done because the covariant derivative of the metric is zero.

```
In[37]:= {gdd[a, b] CovariantD[Au[b], {c, d}], gdd[a, b] AbsoluteD[Au[b], {u, v}]}
         % // MetricSimplify[g]
```

$$\text{Out[37]=}\ \left\{A^{b}{}_{;c\,d}\,g_{a\,b},\ \frac{D^{2}\,A^{b}}{d\,u\,d\,v}\,g_{a\,b}\right\}$$

$$\text{Out[38]=}\ \left\{A_{a;c\,d},\ \frac{D^{2}\,A_{a}}{d\,u\,d\,v}\right\}$$

But MetricSimplify will not raise differentiation indices.

```
In[39]:= guu[a, c] CovariantD[Au[b], {c, d}]
         % // MetricSimplify[g]
```

Out[39]= $A^{b}{}_{;c\,d}\, g^{a\,c}$

Out[40]= $A^{b}{}_{;c\,d}\, g^{a\,c}$

MetricSimplify will not commute with partial or total differentiation because the partial derivative of the metric is not generally zero.

```
In[41]:= gdd[a, b] {PartialD[Au[a], c], PartialD[labs][Au[b], xu[c]], TotalD[Au[a], t]}
         % // MetricSimplify[g]
```

Out[41]= $\left\{A^{a}{}_{,c}\, g_{a\,b},\ g_{a\,b}\, \frac{\partial A^{b}}{\partial x^{c}},\ g_{a\,b}\, \frac{dA^{a}}{dt}\right\}$

Out[42]= $\left\{A^{a}{}_{,c}\, g_{a\,b},\ g_{a\,b}\, \frac{\partial A^{b}}{\partial x^{c}},\ g_{a\,b}\, \frac{dA^{a}}{dt}\right\}$

However, this can be overridden by setting the optional parameter dopartials to True. This might be the case if we are using the flat Minkowski metric or any constant metric.

```
In[43]:= ηdd[a, b] {PartialD[Au[a], c], PartialD[labs][Au[b], xu[c]], TotalD[Au[a], t]}
         % // MetricSimplify[η, True]
```

Out[43]= $\left\{A^{a}{}_{,c}\, \eta_{a\,b},\ \eta_{a\,b}\, \frac{\partial A^{b}}{\partial x^{c}},\ \eta_{a\,b}\, \frac{dA^{a}}{dt}\right\}$

Out[44]= $\left\{A_{b,c},\ \frac{\partial A_{a}}{\partial x^{c}},\ \frac{dA_{b}}{dt}\right\}$

```
In[45]:= ηuu[m, j] PartialD[Sd[m], j] // ToFlavor[red]
         % // MetricSimplify[η, True]
```

Out[45]= $S_{m,j}\, \eta^{m\,j}$

Out[46]= $S^{j}{}_{,j}$

The Kronecker form of the metric will simplify on partial and total derivatives and on differentiation indices.

```
In[47]:= {gud[c, b] PartialD[Au[a], c],
          gud[c, a] PartialD[labs][Au[b], xu[c]], gud[b, a] TotalD[Au[a], t]}
         % // MetricSimplify[g]
```

Out[47]= $\left\{A^{a}{}_{,c}\, g^{c}{}_{b},\ g^{c}{}_{a}\, \frac{\partial A^{b}}{\partial x^{c}},\ g^{b}{}_{a}\, \frac{dA^{a}}{dt}\right\}$

Out[48]= $\left\{A^{a}{}_{,b},\ \frac{\partial A^{b}}{\partial x^{a}},\ \frac{dA^{b}}{dt}\right\}$

Again, MetricSimplify will act on the differentiated tensor, but not on the differentiation coordinate.

```
In[49]:= {ηdd[α, p] ηuu[β, q] PartialD[Au[p], {q, β}],
           ηdd[α, p] ηuu[β, q] PartialD[labs][Au[p], {xu[q], xu[β]}]} // ToFlavor[red]
         % // MetricSimplify[η, True]
```

$$\text{Out[49]}= \left\{A^{p}{}_{,q\,\beta}\,\eta_{\alpha\,p}\,\eta^{\beta\,q},\ \eta_{\alpha\,p}\,\eta^{\beta\,q}\,\frac{\partial^2 A^{p}}{\partial x^{q}\,\partial x^{\beta}}\right\}$$

$$\text{Out[50]}= \left\{A_{\alpha}{}_{,q\,\beta}\,\eta^{\beta\,q},\ \eta^{\beta\,q}\,\frac{\partial^2 A_{\alpha}}{\partial x^{q}\,\partial x^{\beta}}\right\}$$

Double application...

```
In[51]:= guu[i, m] guu[j, n] xd[i] xd[j] // ToFlavor[red]
         % // MetricSimplify[g]
```

$$\text{Out[51]}= g^{i\,m}\,g^{j\,n}\,x_{i}\,x_{j}$$

$$\text{Out[52]}= x^{m}\,x^{n}$$

```
In[53]:= guu[i, m] xd[i] yu[n] + guu[i, m] guu[j, n] xd[i] xd[j] // ToFlavor[red]
         % // MetricSimplify[g]
```

$$\text{Out[53]}= g^{i\,m}\,g^{j\,n}\,x_{i}\,x_{j} + g^{i\,m}\,x_{i}\,y^{n}$$

$$\text{Out[54]}= x^{m}\,x^{n} + x^{m}\,y^{n}$$

We could even have two versions of a metric tensor and simplify selectively.

```
In[55]:= huu[i, m] ηuu[j, n] xd[i] xd[j]
         % // MetricSimplify[h]
         % // MetricSimplify[η]
```

$$\text{Out[55]}= h^{i\,m}\,x_{i}\,x_{j}\,\eta^{j\,n}$$

$$\text{Out[56]}= x^{m}\,x_{j}\,\eta^{j\,n}$$

$$\text{Out[57]}= x^{m}\,x^{n}$$

We can use rules to do partial metric simplifications. We exclude a factor we don't want in the simplification.

```
In[58]:= guu[i, m] guu[j, n] xd[i] yd[j]
         % /. a_. xd[i] :→ xd[i] MetricSimplify[g][a]
```

$$\text{Out[58]}= g^{i\,m}\,g^{j\,n}\,x_{i}\,y_{j}$$

$$\text{Out[59]}= g^{i\,m}\,x_{i}\,y^{n}$$

```
In[60]:= guu[i, m] guu[j, n] xd[i] yd[j]
         % /. a_. guu[j, n] :→ guu[j, n] MetricSimplify[g][a]
```

$$\text{Out[60]}= g^{i\,m}\,g^{j\,n}\,x_{i}\,y_{j}$$

$$\text{Out[61]}= g^{j\,n}\,x^{m}\,y_{j}$$

A better method is to use `MapLevelParts`.

```
In[62]:= guu[i, m] guu[j, n] xd[i] yd[j]
         % // MapLevelParts[MetricSimplify[g], {{1, 3}}]
```

Out[62]= $g^{i\,m}\, g^{j\,n}\, x_i\, y_j$

Out[63]= $g^{j\,n}\, x^m\, y_j$

The following is probably not the metric simplification that is desired. The desired simplification is ambiguous, but Tensorial uses the lowest sort order index as the dummy index and hence lowers the index on x.

```
In[64]:= gdd[i, j] xu[i] yu[j] - xu[i] yd[i]
         % // MetricSimplify[g]
```

Out[64]= $g_{i\,j}\, x^i\, y^j - x^i\, y_i$

Out[65]= $x_j\, y^j - x^i\, y_i$

MapLevelParts will do the simplification we want.

```
In[66]:= gdd[i, j] xu[i] yu[j] - xu[i] yd[i]
         % // MapLevelParts[MetricSimplify[g], {1, {1, 3}}]
```

Out[66]= $g_{i\,j}\, x^i\, y^j - x^i\, y_i$

Out[67]= 0

Or MapLevelPatterns.

```
In[68]:= gdd[i, j] xu[i] yu[j] - xu[i] yd[i]
         % // MapLevelPatterns[MetricSimplify[g], {1, {gdd[i, j] | yu[_]}}]
```

Out[68]= $g_{i\,j}\, x^i\, y^j - x^i\, y_i$

Out[69]= 0

Or we could have done the following...

```
In[70]:= gdd[i, j] xu[i] yu[j] - xu[i] yd[i]
         % // MetricSimplify[g]
         MapAt[UpDownSwap[j], %, 1] // SimplifyTensorSum
```

Out[70]= $g_{i\,j}\, x^i\, y^j - x^i\, y_i$

Out[71]= $x_j\, y^j - x^i\, y_i$

Out[72]= 0

MetricSimplify will work in functions.

```
In[73]:= -I ℏ Exp[gdd[a, b] Au[b] Su[a]] // ToFlavor[red]
         % // MetricSimplify[g]
```

Out[73]= $-i\, e^{A^b\, g_{a\,b}\, S^a}\, \hbar$

Out[74]= $-i\, e^{A_a\, S^a}\, \hbar$

Restore the settings...

```
In[75]:= ClearTensorShortcuts[{{A, S, x, y}, 1}, {{g, h, η, S}, 2}]

In[76]:= DeclareBaseIndices @@ oldindices
         ClearIndexFlavor /@ IndexFlavors;
         DeclareIndexFlavor /@ oldflavors;
         Clear[oldindices, oldflavors, labs]
```

NDim

- NDim gives the current dimension of the underlying linear space for all indices except those flavors that have been assigned special base index sets.

NDim is set by DeclareBaseIndices. It is the length of the BaseIndices list, which applies to all indices except those flavors that have been assigned special base index sets..

On loading, Tensorial automatically initializes the base indices to {1, 2, 3} and so NDim == 3.

See also: DeclareBaseIndices, BaseIndices, CompleteBaseIndices, BaseIndexQ.

Examples

```
In[1]:= Needs["TensorCalculus4`Tensorial`"]

In[2]:= oldindices = CompleteBaseIndices

Out[2]= {{1, 2, 3}}
```

The following gives the current dimension and set of base indices...

```
In[3]:= {NDim, BaseIndices}

Out[3]= {3, {1, 2, 3}}
```

The following statement gives a new set of base indices and a new value for NDim.

```
In[4]:= DeclareBaseIndices[Range[0, 3]]
        {NDim, BaseIndices}

Out[5]= {4, {0, 1, 2, 3}}
```

This resets to the original indices.

```
In[6]:= DeclareBaseIndices @@ oldindices
        {NDim, BaseIndices}
        Clear[oldindices]

Out[7]= {3, {1, 2, 3}}
```

NondependentPartialD

- `NondependentPartialD[{lab1, lab2, ...} ...][expr]` will implement the assumption that Tensor labels, {lab1,lab2,...} are nondependent so that a partial derivative of one with respect to another will be zero.

Multiple lists of nondependent Tensor labels may be provided.

See also: `PartialD`, `ExpandPartialD`, `TotalD`, `CovariantD`, `AbsoluteD`.

Examples

```
In[1]:= Needs["TensorCalculus4`Tensorial`"]

In[2]:= oldbasis = CompleteBaseIndices;

In[3]:= DefineTensorShortcuts[{{x, y, z, f}, 1}, {{η, δ}, 2}]
        labs = {x, δ, g, Γ};
```

Here is an expression involving partial differentials.

```
In[5]:= expr = PartialD[labs][yu[i], Tensor /@ {ϕ, ψ}] + PartialD[labs][xu[i], zu[j]] +
          PartialD[labs][zu[i], xu[j]] + PartialD[labs][yu[i], xu[j]]
```

$$\text{Out[5]}= \frac{\partial x^i}{\partial z^j} + \frac{\partial^2 y^i}{\partial \phi\, \partial \psi} + \frac{\partial y^i}{\partial x^j} + \frac{\partial z^i}{\partial x^j}$$

Here we specify that y and ϕ are independent. Notice that y is still dependent on x so that term is not zeroed.

```
In[6]:= expr
        % // NondependentPartialD[{y, ϕ}]
```

$$\text{Out[6]}= \frac{\partial x^i}{\partial z^j} + \frac{\partial^2 y^i}{\partial \phi\, \partial \psi} + \frac{\partial y^i}{\partial x^j} + \frac{\partial z^i}{\partial x^j}$$

$$\text{Out[7]}= \frac{\partial x^i}{\partial z^j} + \frac{\partial y^i}{\partial x^j} + \frac{\partial z^i}{\partial x^j}$$

Here we specify that y is independent of x but not of ϕ or ψ.

```
In[8]:= expr
        % // NondependentPartialD[{y, x}]
```

$$\text{Out[8]}= \frac{\partial x^i}{\partial z^j} + \frac{\partial^2 y^i}{\partial \phi\, \partial \psi} + \frac{\partial y^i}{\partial x^j} + \frac{\partial z^i}{\partial x^j}$$

$$\text{Out[9]}= \frac{\partial x^i}{\partial z^j} + \frac{\partial^2 y^i}{\partial \phi\, \partial \psi} + \frac{\partial z^i}{\partial x^j}$$

Here we specify that x and z are independent. Both partial derivatives are eliminated.

```
In[10]:= expr
         % // NondependentPartialD[{x, z}]
```

$$\text{Out[10]}= \frac{\partial x^i}{\partial z^j} + \frac{\partial^2 y^i}{\partial \phi\, \partial \psi} + \frac{\partial y^i}{\partial x^j} + \frac{\partial z^i}{\partial x^j}$$

$$\text{Out[11]}= \frac{\partial^2 y^i}{\partial \phi\, \partial \psi} + \frac{\partial y^i}{\partial x^j}$$

Here all the variables are independent...

```
In[12]:= expr
         % // NondependentPartialD[{x, y, z, ϕ, ψ}]
```

$$\text{Out[12]}= \frac{\partial x^i}{\partial z^j} + \frac{\partial^2 y^i}{\partial \phi\, \partial \psi} + \frac{\partial y^i}{\partial x^j} + \frac{\partial z^i}{\partial x^j}$$

```
Out[13]= 0
```

Sometimes we want derivatives of certain tensors to be zero. We illustrate this by writing a routine that will derive the scalar Euler-Lagrange equation from the Lagrangian for the Klein-Gordon equation.

```
In[14]:= DeclareBaseIndices[{0, 1, 2, 3}]
         SetTensorValueRules[#, DiagonalMatrix[{-1, 1, 1, 1}]] & /@ {ηdd[a, b], ηuu[a, b]};
         labs = {x, δ, η, Γ};
```

The real Klein-Gordon Lagrangian is:

```
In[17]:= RKGL =
          1 / 2 PartialD[labs][Tensor[ϕ], xu[μ]] * PartialD[labs][Tensor[ϕ], xu[ν]] ηuu[μ, ν] -
           m^2 / 2 * Tensor[ϕ]^2
```

$$\text{Out[17]}= -\frac{1}{2} m^2 (\phi)^2 + \frac{1}{2} \eta^{\mu\nu} \frac{\partial \phi}{\partial x^\mu} \frac{\partial \phi}{\partial x^\nu}$$

Where m is a constant

```
In[18]:= Attributes[m] = {Constant};
```

But it is better to make a variable change

```
In[19]:= RKGLmod =
          RKGL /. PartialD[labs][Tensor[ϕ], xu[ν_] ] → fd[ν]
```

$$\text{Out[19]}= -\frac{1}{2} m^2 (\phi)^2 + \frac{1}{2} f_\mu f_\nu \eta^{\mu\nu}$$

We can then calculate the Euler-Lagrange equation by using the fact that $\{\eta, x\}$ are independent and $\{\eta, f, \phi\}$ are independent.

```
In[20]:= Print["Modified Lagrangian..."]
         RKGLmod
         Print["Euler-Lagrange equations ∂x ∂f (L) - ∂ϕ(L)"]
         PartialD[labs][PartialD[{f, δ, η, Γ}][RKGLmod, fd[r]], xu[r]] -
          PartialD[{ϕ, δ, g, Γ}][RKGLmod, Tensor[ϕ]]
         Print["Eliminating nondependencies..."]
         %% // NondependentPartialD[{η, x}, {η, f, ϕ}]
         Print["Absorbing the Kronecker."]
         %% // KroneckerAbsorb[δ]
         Print["Making the reverse substitution for the f's."]
         %% /. fd[v_] → PartialD[labs][Tensor[ϕ], xu[v]]
         Print["Expanding."]
         %% // ToArrayValues[]
```

```
Modified Lagrangian...
```

$$\text{Out[21]}= -\frac{1}{2}\,m^2\,(\phi)^2+\frac{1}{2}\,f_\mu\,f_\nu\,\eta^{\mu\nu}$$

```
Euler-Lagrange equations ∂x ∂f (L) - ∂ϕ(L)
```

$$\text{Out[23]}= m^2\,\phi - m^2\left(\frac{\partial\phi}{\partial f_r}\,\frac{\partial\phi}{\partial x^r}+\phi\,\frac{\partial\,\frac{\partial\phi}{\partial f_r}}{\partial x^r}\right)+\frac{1}{2}\left(\left(\delta^r{}_\nu\,\eta^{\mu\nu}+f_\nu\,\frac{\partial\eta^{\mu\nu}}{\partial f_r}\right)\frac{\partial f_\mu}{\partial x^r}+\delta^r{}_\mu\,\eta^{\mu\nu}\,\frac{\partial f_\nu}{\partial x^r}+f_\nu\,\delta^r{}_\mu\,\frac{\partial\eta^{\mu\nu}}{\partial x^r}+\right.$$
$$\left.f_\mu\left(\frac{\partial\eta^{\mu\nu}}{\partial f_r}\,\frac{\partial f_\nu}{\partial x^r}+\delta^r{}_\nu\,\frac{\partial\eta^{\mu\nu}}{\partial x^r}+f_\nu\,\frac{\partial\,\frac{\partial\eta^{\mu\nu}}{\partial f_r}}{\partial x^r}\right)\right)+\frac{1}{2}\left(-f_\nu\,\eta^{\mu\nu}\,\frac{\partial f_\mu}{\partial\phi}-f_\mu\left(\eta^{\mu\nu}\,\frac{\partial f_\nu}{\partial\phi}+f_\nu\,\frac{\partial\eta^{\mu\nu}}{\partial\phi}\right)\right)$$

```
Eliminating nondependencies...
```

$$\text{Out[25]}= m^2\,\phi+\frac{1}{2}\left(\delta^r{}_\nu\,\eta^{\mu\nu}\,\frac{\partial f_\mu}{\partial x^r}+\delta^r{}_\mu\,\eta^{\mu\nu}\,\frac{\partial f_\nu}{\partial x^r}\right)$$

```
Absorbing the Kronecker.
```

$$\text{Out[27]}= m^2\,\phi+\frac{1}{2}\left(\eta^{\mu\nu}\,\frac{\partial f_\mu}{\partial x^\nu}+\eta^{\mu\nu}\,\frac{\partial f_\nu}{\partial x^\mu}\right)$$

```
Making the reverse substitution for the f's.
```

$$\text{Out[29]}= m^2\,\phi+\frac{1}{2}\left(\eta^{\mu\nu}\,\frac{\partial^2\phi}{\partial x^\mu\,\partial x^\nu}+\eta^{\mu\nu}\,\frac{\partial^2\phi}{\partial x^\nu\,\partial x^\mu}\right)$$

```
Expanding.
```

$$\text{Out[31]}= m^2\,\phi-\frac{\partial^2\phi}{\partial x^0\,\partial x^0}+\frac{\partial^2\phi}{\partial x^1\,\partial x^1}+\frac{\partial^2\phi}{\partial x^2\,\partial x^2}+\frac{\partial^2\phi}{\partial x^3\,\partial x^3}$$

Restore settings.

```
In[32]:= ClearTensorValues /@ {ηdd[i, j], ηuu[i, j]};
         ClearTensorShortcuts[{{x, y, z, f}, 1}, {{η, δ}, 2}]
         ClearAll[m];

In[35]:= DeclareBaseIndices @@ oldbasis
         Clear[oldindices, labs, RKGL, RKGLmod]
```

NonzeroValueRules

- `NonzeroValueRules[label]` gives only the rules for the label that have nonzero right hand sides.
- `NonzeroValueRules[label1, label2, ...]` concatenates the nonzero rules for several labels.

This is useful for viewing the values for tensors and objects that have many zero values.

SelectedTensorRules is even better at selecting display sets.

`NonzeroValueRules` does not simplify the right hand sides and checks only for explicit zero values. Values rules should be simplified as much as possible before they are stored and before `NonzeroValueRules` is used.

`NonzeroValueRules` should not be used for substitution as it will not substitute zero values!

See also: `SelectedTensorRules`, `SetTensorValueRules`, `SetTensorValues`, `ClearTensorValues`, `Tensor`, `UseCoordinates`.

Examples

```
In[1]:= Needs["TensorCalculus4`Tensorial`"]
```

Save the settings.

```
In[2]:= oldindices = CompleteBaseIndices;
        oldflavors = IndexFlavors;
        ClearIndexFlavor /@ oldflavors;
        DeclareIndexFlavor[{red, Red}]
```

We will set up the metric for a wormhole in general relativity and work in a red flavor. (From an excellent book *Gravity* by James B. Hartle.)

```
In[6]:= DeclareBaseIndices[{t, r, θ, ϕ}]
        DefineTensorShortcuts[{x, 1}, {g, 2}, {Γ, 3}]
        labs = {x, δ, g, Γ};
```

```
In[9]:= SetAttributes[b, Constant];
        (cmetric = DiagonalMatrix[{-1, 1, b^2 + r^2, (b^2 + r^2) Sin[θ]^2}]) // MatrixForm
        (metric = cmetric // CoordinatesToTensors[{t, r, θ, ϕ}, x, red]) // MatrixForm
        MapThread[SetTensorValueRules[#1, #2] &,
          {{gdd[a, b], guu[a, b]} // ToFlavor[red], {metric, Inverse@metric}}];
```

```
Out[10]//MatrixForm=
⎛-1  0  0        0                  ⎞
⎜0   1  0        0                  ⎟
⎜0   0  b^2 + r^2  0                ⎟
⎝0   0  0        (b^2 + r^2) Sin[θ]^2⎠
```

```
Out[11]//MatrixForm=
⎛-1  0  0            0                      ⎞
⎜0   1  0            0                      ⎟
⎜0   0  b^2 + (x^r)^2  0                    ⎟
⎝0   0  0            Sin[x^θ]^2 (b^2 + (x^r)^2)⎠
```

Next we calculate and set rules for the Christoffel symbols.

```
In[13]:= MapThread[SetTensorValueRules[#1, #2] &,
           {{Γddd[a, b, c], Γudd[a, b, c]} // ToFlavor[red],
            CalculateChristoffels[labs, red, Simplify[#, Trig → False] &]}];
```

Using `NonzeroValueRules` we can obtain a short list of the Christoffel symbols. `TensorValueRules` would have given us a much longer list.

```
In[14]:= NonzeroValueRules[Γ] // UseCoordinates[{t, r, θ, ϕ}, x, red] // TableForm
```

```
Out[14]//TableForm=
Γ_{r θ θ} → -r
Γ_{r ϕ ϕ} → -r Sin[θ]^2
Γ_{θ r θ} → r
Γ_{θ θ r} → r
Γ_{θ ϕ ϕ} → -(b^2 + r^2) Cos[θ] Sin[θ]
Γ_{ϕ r ϕ} → r Sin[θ]^2
Γ_{ϕ θ ϕ} → (b^2 + r^2) Cos[θ] Sin[θ]
Γ_{ϕ ϕ r} → r Sin[θ]^2
Γ_{ϕ ϕ θ} → (b^2 + r^2) Cos[θ] Sin[θ]
Γ^r_{θ θ} → -r
Γ^r_{ϕ ϕ} → -r Sin[θ]^2
Γ^θ_{r θ} → r/(b^2+r^2)
Γ^θ_{θ r} → r/(b^2+r^2)
Γ^θ_{ϕ ϕ} → -Cos[θ] Sin[θ]
Γ^ϕ_{r ϕ} → r/(b^2+r^2)
Γ^ϕ_{θ ϕ} → Cot[θ]
Γ^ϕ_{ϕ r} → r/(b^2+r^2)
Γ^ϕ_{ϕ θ} → Cot[θ]
```

Restore settings.

```
In[15]:= ClearTensorValues /@
           ToFlavor[red] /@ {gdd[a, b], guu[a, b], Γudd[i, j, k], Γddd[i, j, k]};
         ClearTensorShortcuts[{x, 1}, {g, 2}, {Γ, 3}]
```

```
In[17]:= DeclareBaseIndices @@ oldindices
         ClearIndexFlavor /@ IndexFlavors;
         DeclareIndexFlavor[oldflavors];
         Clear[oldindices, oldflavors, labs]
```

OrthonormalTransformation

- `OrthonormalTransformation[`*metric, signature, simplifyroutine, samplevalues*`]` will return a transformation matrix from the coordinate basis for metric to an orthonormal basis where the metric takes the form of a diagonal matrix with the signature along the diagonal.

The signature must consist of NDim length vector with entries of +1 or -1.

If metric is diagonal to begin with, the routine simply calculates scaling factors so the resulting basis will be normalized. In this case the new basis vectors will have a correspondence with the old basis vectors.

If the metric is not diagonal an eigensystem calculation is performed and the new basis vectors will have no simple correspondence with the old basis vectors. You may wish to use an additional permutation matrix to establish a partial correspondence for basis vectors that have a single entry.

The optional simplify routine will be used to simplify elements in the calculation. In particular the routine tries to establish the signs of the diagonal elements or eigenvalues. A simplifyroutine that contains assumptions on the domains of various parameters or coordinates will often be necessary. Samplevalues is another optional argument that provides sample values in the form of substitution rules

Samplevalues is another optional argument that provides sample values in the form of substitution rules. It is a fallback method for calculating the signs of the elements in case the simplifyroutine cannot evaluate the Sign expressions to values.

It will usually be more convenient to use the metric with symbols for coordinates, instead of the indexed coordinates, and then convert the resulting matrix to indexed coordinates afterwards.

See also: `ChristoffelFromGeodesics`, `CalculateChristoffels`.

Example - Schwarzschild Riemann Components in Orthonormal Basis

```
In[1]:= Needs["TensorCalculus4`Tensorial`"]

In[2]:= oldindices = CompleteBaseIndices;
        oldflavors = IndexFlavors;
```

We will calculate an orthonormal basis for the Schwarzschild metric and then express the Riemann tensor in the orthonormal basis. This reproduces the calculation in Appendix B of James Hartle's *Gravitation*, p546.

```
In[4]:= varnames = {t, r, θ, ϕ};
        DeclareBaseIndices[varnames]
        DeclareIndexFlavor[{red, Red}]
        labs = {x, δ, g, Γ};
        DefineTensorShortcuts[{{x, e}, 1}, {{g, δ, G, L}, 2}, {Γ, 3}, {R, 4}]
        SetTensorValues[δud[μ, ν], IdentityMatrix[4]]
```

The metric matrix is...

```
In[10]:= SetAttributes[M, Constant];
         (cmetric = DiagonalMatrix[
             {-(1 - 2 M / r), (1 - 2 M / r)^-1, r^2 {1, Sin[θ]^2}}] // Flatten]) // MatrixForm
         metric = % // CoordinatesToTensors[varnames];
         MapThread[SetTensorValues[#1, #2] &,
           {{gdd[a, b], guu[a, b]}, {metric, Inverse@metric}}];
```

Out[11]//MatrixForm=

$$\begin{pmatrix} -1+\frac{2M}{r} & 0 & 0 & 0 \\ 0 & \frac{1}{1-\frac{2M}{r}} & 0 & 0 \\ 0 & 0 & r^2 & 0 \\ 0 & 0 & 0 & r^2 \sin[\theta]^2 \end{pmatrix}$$

Since the metric is diagonal the simpler scaling algorithm is used. We will calculate with the metric in symbolic coordinate form.

```
In[14]:= (Lmatrix = OrthonormalTransformation[cmetric,
             {-1, 1, 1, 1}, Simplify[#, M > 0 ∧ r > 2 M ∧ 0 < θ < π] &]) // MatrixForm
         % // CoordinatesToTensors[varnames];
         SetTensorValues[Lud[a, red@α], %]
```

Out[14]//MatrixForm=

$$\begin{pmatrix} \frac{1}{\sqrt{1-\frac{2M}{r}}} & 0 & 0 & 0 \\ 0 & \sqrt{1-\frac{2M}{r}} & 0 & 0 \\ 0 & 0 & \frac{1}{r} & 0 \\ 0 & 0 & 0 & \frac{\text{Csc}[\theta]}{r} \end{pmatrix}$$

The orthonormal basis vectors, with red indices, in terms of the coordinate basis vectors are then...

```
In[17]:= ed[red@α] == Lud[a, red@α] ed[a]
         % // ToArrayValues[] // UseCoordinates[varnames] // TableForm
```

Out[17]= $e_\alpha == e_a \, L^a{}_\alpha$

Out[18]//TableForm=

$$e_t == \frac{e_t}{\sqrt{1-\frac{2M}{r}}}$$

$$e_r == \sqrt{1-\frac{2M}{r}}\, e_r$$

$$e_\theta == \frac{e_\theta}{r}$$

$$e_\phi == \frac{\text{Csc}[\theta]\, e_\phi}{r}$$

We can check that the new basis vectors are orthonormal.

```
In[19]:= Print["Check the orthogonality of the red basis vectors"]
         Outer[ed[#1].ed[#2] &, red /@ BaseIndices, red /@ BaseIndices] // MatrixForm
         Print["Substitute in terms of the coordinate basis"]
         %% /. ed[red@α_].ed[red@β_] → (Lud[a, red@α] ed[a]).(Lud[b, red@β] ed[b]) //
          MatrixForm
         Print["Evaluate the dot products"]
         %% // EvaluateDotProducts[e, g, False] // MatrixForm
         Print["Perform the summations and simplify"]
         %% // EinsteinSum[] // Simplify // MatrixForm
```

Check the orthogonality of the red basis vectors

Out[20]//MatrixForm=

$$\begin{pmatrix} e_t.e_t & e_t.e_r & e_t.e_\theta & e_t.e_\phi \\ e_r.e_t & e_r.e_r & e_r.e_\theta & e_r.e_\phi \\ e_\theta.e_t & e_\theta.e_r & e_\theta.e_\theta & e_\theta.e_\phi \\ e_\phi.e_t & e_\phi.e_r & e_\phi.e_\theta & e_\phi.e_\phi \end{pmatrix}$$

Substitute in terms of the coordinate basis

Out[22]//MatrixForm=

$$\begin{pmatrix} (e_a L^a{}_t).(e_b L^b{}_t) & (e_a L^a{}_t).(e_b L^b{}_r) & (e_a L^a{}_t).(e_b L^b{}_\theta) & (e_a L^a{}_t).(e_b L^b{}_\phi) \\ (e_a L^a{}_r).(e_b L^b{}_t) & (e_a L^a{}_r).(e_b L^b{}_r) & (e_a L^a{}_r).(e_b L^b{}_\theta) & (e_a L^a{}_r).(e_b L^b{}_\phi) \\ (e_a L^a{}_\theta).(e_b L^b{}_t) & (e_a L^a{}_\theta).(e_b L^b{}_r) & (e_a L^a{}_\theta).(e_b L^b{}_\theta) & (e_a L^a{}_\theta).(e_b L^b{}_\phi) \\ (e_a L^a{}_\phi).(e_b L^b{}_t) & (e_a L^a{}_\phi).(e_b L^b{}_r) & (e_a L^a{}_\phi).(e_b L^b{}_\theta) & (e_a L^a{}_\phi).(e_b L^b{}_\phi) \end{pmatrix}$$

Evaluate the dot products

Out[24]//MatrixForm=

$$\begin{pmatrix} g_{ab} L^a{}_t L^b{}_t & g_{ab} L^a{}_t L^b{}_r & g_{ab} L^a{}_t L^b{}_\theta & g_{ab} L^a{}_t L^b{}_\phi \\ g_{ab} L^a{}_r L^b{}_t & g_{ab} L^a{}_r L^b{}_r & g_{ab} L^a{}_r L^b{}_\theta & g_{ab} L^a{}_r L^b{}_\phi \\ g_{ab} L^a{}_\theta L^b{}_t & g_{ab} L^a{}_\theta L^b{}_r & g_{ab} L^a{}_\theta L^b{}_\theta & g_{ab} L^a{}_\theta L^b{}_\phi \\ g_{ab} L^a{}_\phi L^b{}_t & g_{ab} L^a{}_\phi L^b{}_r & g_{ab} L^a{}_\phi L^b{}_\theta & g_{ab} L^a{}_\phi L^b{}_\phi \end{pmatrix}$$

Perform the summations and simplify

Out[26]//MatrixForm=

$$\begin{pmatrix} -1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

So indeed the new basis vectors are orthonormal and give us the Minkowski metric matrix.

If we had not given enough information to fully simplify we would have obtained a warning message and a less than simplified result.

```
In[27]:= OrthonormalTransformation[cmetric, {-1, 1, 1, 1}, Simplify]
```

```
OrthonormalTransformation::signs :
 Warning, signs of eigenvalues {Sign[-1 + 2 M/r], 1/Sign[1 - 2 M/r],
    Sign[r]^2, Sign[r]^2 Sign[Sin[θ]]^2} do not match signature {-1, 1, 1, 1}
  or are undetermined. Permutation matrix omitted for general case.
```

Out[27]= $$\left\{\left\{\frac{1}{\sqrt{1-\frac{2M}{r}}}, 0, 0, 0\right\}, \left\{0, \frac{1}{\sqrt{\frac{1}{1-\frac{2M}{r}}}}, 0, 0\right\}, \left\{0, 0, \frac{1}{\sqrt{r^2}}, 0\right\}, \left\{0, 0, 0, \frac{1}{\sqrt{r^2 \text{Sin}[\theta]^2}}\right\}\right\}$$

If we had given sample value rules, we would have eliminated the warning message but still obtained a less than simplified result.

```
In[28]:= OrthonormalTransformation[cmetric, {-1, 1, 1, 1}, Simplify, {M → 1, r → 3, θ → π / 2}]
```

$$\text{Out[28]= } \left\{\left\{\frac{1}{\sqrt{1-\frac{2M}{r}}}, 0, 0, 0\right\}, \left\{0, \frac{1}{\sqrt{\frac{1}{1-\frac{2M}{r}}}}, 0, 0\right\}, \left\{0, 0, \frac{1}{\sqrt{r^2}}, 0\right\}, \left\{0, 0, 0, \frac{1}{\sqrt{r^2\,\text{Sin}[\theta]^2}}\right\}\right\}$$

To calculate the orthonormal Riemann components we first calculate the Christoffel connections. First we set the values as rules so we can easily look at them, and then reset them as direct values.

```
In[29]:= MapThread[SetTensorValueRules[#1, #2] &,
           {{Γddd[a, b, c], Γudd[a, b, c]}, {Γdown, Γup} =
             CalculateChristoffels[labs, Identity, Simplify[#, Trig → False] &]}];
         SelectedTensorRules[Γ, Γudd[_, j_, k_] /; OrderedQ[{j, k}]] //
           UseCoordinates[varnames] // TableForm
         MapThread[SetTensorValues[#1, #2] &,
           {{Γddd[a, b, c], Γudd[a, b, c]}, {Γdown, Γup} = CalculateChristoffels[labs]}];
```

Out[30]//TableForm=

$$\Gamma^{t}{}_{r\,t} \to -\frac{M}{2Mr-r^2}$$
$$\Gamma^{r}{}_{t\,t} \to \frac{M(-2M+r)}{r^3}$$
$$\Gamma^{r}{}_{r\,r} \to \frac{M}{2Mr-r^2}$$
$$\Gamma^{r}{}_{\theta\,\theta} \to 2M-r$$
$$\Gamma^{r}{}_{\phi\,\phi} \to (2M-r)\,\text{Sin}[\theta]^2$$
$$\Gamma^{\theta}{}_{r\,\theta} \to \frac{1}{r}$$
$$\Gamma^{\theta}{}_{\phi\,\phi} \to -\text{Cos}[\theta]\,\text{Sin}[\theta]$$
$$\Gamma^{\phi}{}_{r\,\phi} \to \frac{1}{r}$$
$$\Gamma^{\phi}{}_{\theta\,\phi} \to \text{Cot}[\theta]$$

We then calculate the down Riemann tensor in the coordinate basis.

```
In[32]:= riemannd = CalculateRiemannd[labs, Identity, Simplify[#, Trig → False] &];
         SetTensorValueRules[Rdddd[μ, ν, ρ, σ], riemannd]
         (SelectedTensorRules[R, Rdddd[a_, b_, c_, d_] /; OrderedQ[{a, b}] ∧
               OrderedQ[{c, d}] ∧ OrderedQ[{{a, b}, {c, d}}] ∧ Head[a] === Symbol] //
           UseCoordinates[varnames]) // TableForm
         SetTensorValues[Rdddd[μ, ν, ρ, σ], riemannd]
```

Out[34]//TableForm=

$$R_{t\,\theta\,t\,\theta} \to \frac{M(-2M+r)}{r^2}$$
$$R_{t\,\phi\,t\,\phi} \to \frac{M(-2M+r)\,\text{Sin}[\theta]^2}{r^2}$$
$$R_{r\,t\,r\,t} \to -\frac{2M}{r^3}$$
$$R_{r\,\theta\,r\,\theta} \to \frac{M}{2M-r}$$
$$R_{r\,\phi\,r\,\phi} \to \frac{M\,\text{Sin}[\theta]^2}{2M-r}$$
$$R_{\theta\,\phi\,\theta\,\phi} \to 2Mr\,\text{Sin}[\theta]^2$$

We will first calculate the orthonormal Riemann components using array multiplication. The following shows how this is done using a Tensorial dot mode calculation. The array output is actually supressed because the array is so large.

```
In[36]:= Print["Transformation of Riemann tensor to the orthonormal basis."]
         Lud[a, red@α] Lud[b, red@β] Lud[c, red@γ] Lud[d, red@δ] Rdddd[a, b, c, d]
         Print["Going to dot mode."]
         %% // DotTensorFactors[{5, 4, 3, 2, 1}]
         Print["We have to keep multiplying the first two arrays and then transposing the
            resulting array by {2,3,4,1}, which shifts the next index into position."]
         %% // ExpandDotArray[Tensor[R, _, _]];
         (step1 = Fold[(#1 // ExpandDotArray[Lud[#2, _]] // DotOperate[1, Function[{A, B},
                   Transpose[A.B, {2, 3, 4, 1}]]]) &, %, {d, c, b, a}];) // Timing
         Print["Set and display the independent components."]
         SetTensorValueRules[
          Rdddd[μ, ν, ρ, σ] // ToFlavor[red], step1 /. MatrixForm → Identity]
         (SelectedTensorRules[R, Rdddd[a_, b_, c_, d_] /;
               OrderedQ[{a, b}] ∧ OrderedQ[{c, d}] ∧ OrderedQ[{{a, b}, {c, d}}]] //
            UseCoordinates[varnames] // Simplify) // TableForm
```

Transformation of Riemann tensor to the orthonormal basis.

$$\text{Out[37]=}\quad L^{a}{}_{\alpha}\, L^{b}{}_{\beta}\, L^{c}{}_{\gamma}\, L^{d}{}_{\delta}\, R_{abcd}$$

Going to dot mode.

$$\text{Out[39]=}\quad R_{abcd}.L^{d}{}_{\delta}.L^{c}{}_{\gamma}.L^{b}{}_{\beta}.L^{a}{}_{\alpha}$$

We have to keep multiplying the first two arrays and then transposing the resulting array by {2,3,4,1}, which shifts the next index into position.

$$\text{Out[42]=}\quad \{0.093\ \text{Second}, \text{Null}\}$$

Set and display the independent components.

Out[45]//TableForm=

$$R_{t\theta t\theta} \to \frac{M}{r^3}$$
$$R_{t\phi t\phi} \to \frac{M}{r^3}$$
$$R_{rtrt} \to -\frac{2M}{r^3}$$
$$R_{r\theta r\theta} \to -\frac{M}{r^3}$$
$$R_{r\phi r\phi} \to -\frac{M}{r^3}$$
$$R_{\theta\phi\theta\phi} \to \frac{2M}{r^3}$$

Using Tensorial array expansion takes longer than the array multiplication method.

```
In[46]:= Lud[a, red@α] Lud[b, red@β] Lud[c, red@γ] Lud[d, red@δ] Rdddd[a, b, c, d]
         (onriemannd = % // ToArrayValues[];) // Timing
         SetTensorValueRules[Rdddd[μ, ν, ρ, σ] // ToFlavor[red], onriemannd]
         (SelectedTensorRules[R, Rdddd[a_, b_, c_, d_] /;
               OrderedQ[{a, b}] ∧ OrderedQ[{c, d}] ∧ OrderedQ[{{a, b}, {c, d}}]] //
           UseCoordinates[varnames] // Simplify) // TableForm
```

$$\text{Out[46]= } L^{a}{}_{\alpha}\, L^{b}{}_{\beta}\, L^{c}{}_{\gamma}\, L^{d}{}_{\delta}\, R_{abcd}$$

$$\text{Out[47]= } \{1.516\ \text{Second}, \text{Null}\}$$

Out[49]//TableForm=

$$R_{t\theta t\theta} \to \frac{M}{r^3}$$
$$R_{t\phi t\phi} \to \frac{M}{r^3}$$
$$R_{rtrt} \to -\frac{2M}{r^3}$$
$$R_{r\theta r\theta} \to -\frac{M}{r^3}$$
$$R_{r\phi r\phi} \to -\frac{M}{r^3}$$
$$R_{\theta\phi\theta\phi} \to \frac{2M}{r^3}$$

Restore state

```
In[50]:= DeclareBaseIndices @@ oldindices
         DeclareIndexFlavor @@ oldflavors;

In[52]:= Clear[Lmatrix, metric, cmetric, riemannd, criemannd, oldindices, oldflavors]
         {gdd[i, j], guu[i, j], Rdddd[i, j, k, l],
          Rdddd @@ red /@ {i, j, k, l}, Γddd[i, j, k], Γudd[i, j, k]}
         ClearTensorValues[% // Evaluate]
```

$$\text{Out[53]= } \left\{g_{ij},\ g^{ij},\ R_{ijkl},\ R_{ijkl},\ \Gamma_{ijk},\ \Gamma^{i}{}_{jk}\right\}$$

```
In[55]:= ClearTensorShortcuts[{{x, e}, 1}, {{g, δ, G, L}, 2}, {Γ, 3}, {R, 4}]
```

Example - Orthonormal Basis for Painlevé-Gullstrand Metric

The Painlevé-Gullstrand metric is an alternative to the Schwarzschild metric. It has off diagonal terms and hence invokes the more general algorithm for converting to an orthonormal basis.

```
In[56]:= Needs["TensorCalculus4`Tensorial`"]

In[57]:= oldindices = CompleteBaseIndices;
         oldflavors = IndexFlavors;
```

```
In[59]:= DeclareBaseIndices[{0, 1, 2, 3}]
         varnames = {u, r, θ, ϕ};
         DeclareIndexFlavor[{red, Red}]
         labs = {x, δ, g, Γ};
         DefineTensorShortcuts[{{x, e}, 1}, {{g, δ, G, L}, 2}, {Γ, 3}, {R, 4}]
         SetTensorValues[δud[i, j], IdentityMatrix[NDim]]

         SetAttributes[M, Constant];

         cmetric = ( 1 - 2M/r   -√(2M/r)    0        0
                   -√(2M/r)       -1        0        0
                       0           0      -r^2       0
                       0           0       0   -r^2 Sin[θ]^2 );

         metric = cmetric // CoordinatesToTensors[{u, r, θ, ϕ}];
         MapThread[SetTensorValues[#1, #2] &,
           {{gdd[a, b], guu[a, b]}, {metric, Inverse@metric}}];
```

We will calculate with the metric in symbolic coordinate form. The timelike form of the metric has been used. The resulting matrix is fairly complicated algebrically.

```
In[69]:= Lmatrix = OrthonormalTransformation[cmetric,
           {1, -1, -1, -1}, FullSimplify[#, M > 0 ∧ r > 0 ∧ 0 < θ < π] &]
         % // CoordinatesToTensors[varnames];
         SetTensorValues[Lud[a, red@α], %]
```

$$\text{Out[69]}= \left\{\left\{-\frac{\sqrt{M+r+\sqrt{M^2+r^2}}}{\sqrt{2}\ (M^2+r^2)^{1/4}},\ 0,\ \frac{\sqrt{M+r-\sqrt{M^2+r^2}}}{\sqrt{2}\ (M^2+r^2)^{1/4}},\ 0\right\},\right.$$
$$\left\{\sqrt{\frac{r}{\left(-M+\sqrt{M^2+r^2}\right)\left(1+\frac{\left(-M+r+\sqrt{M^2+r^2}\right)^2}{2Mr}\right)}},\ 0,\ \sqrt{\frac{r}{\left(M+\sqrt{M^2+r^2}\right)\left(1+\frac{\left(M-r+\sqrt{M^2+r^2}\right)^2}{2Mr}\right)}},\ 0\right\},$$
$$\left.\left\{0,\ \frac{1}{r},\ 0,\ 0\right\},\ \left\{0,\ 0,\ 0,\ \frac{\text{Csc}[\theta]}{r}\right\}\right\}$$

The orthonormal basis vectors, with red indices, in terms of the coordinate basis vectors are then...

```
In[72]:= ed[red@α] == Lud[a, red@α] ed[a]
         % // ToArrayValues[] // UseCoordinates[varnames] // TableForm
```

$$\text{Out[72]}= e_\alpha == e_a\ L^a{}_\alpha$$

Out[73]//TableForm=

$$e_0 == -\frac{\sqrt{M+r+\sqrt{M^2+r^2}}}{\sqrt{2}\ (M^2+r^2)^{1/4}}\ e_0 + \sqrt{\frac{r}{\left(-M+\sqrt{M^2+r^2}\right)\left(1+\frac{\left(-M+r+\sqrt{M^2+r^2}\right)^2}{2Mr}\right)}}\ e_1$$

$$e_1 == \frac{e_2}{r}$$

$$e_2 == \frac{\sqrt{M+r-\sqrt{M^2+r^2}}}{\sqrt{2}\ (M^2+r^2)^{1/4}}\ e_0 + \sqrt{\frac{r}{\left(M+\sqrt{M^2+r^2}\right)\left(1+\frac{\left(M-r+\sqrt{M^2+r^2}\right)^2}{2Mr}\right)}}\ e_1$$

$$e_3 == \frac{\text{Csc}[\theta]\ e_3}{r}$$

We would prefer to switch the order of the 1 and 2 red basis vectors so that the indices of the unmixed basis vectors correspond to the same indices of the coordinate basis vectors. This is done by multiplying Lmatrix by a permutation matrix.

```
In[74]:= Lmatrix = Lmatrix.Part[IdentityMatrix[NDim], {1, 3, 2, 4}]
         % // CoordinatesToTensors[varnames];
         SetTensorValues[Lud[a, red@α], %]
```

$$\text{Out[74]}= \left\{\left\{-\frac{\sqrt{M+r+\sqrt{M^2+r^2}}}{\sqrt{2}\ (M^2+r^2)^{1/4}},\ \frac{\sqrt{M+r-\sqrt{M^2+r^2}}}{\sqrt{2}\ (M^2+r^2)^{1/4}},\ 0,\ 0\right\},\right.$$
$$\left\{\sqrt{\frac{r}{\left(-M+\sqrt{M^2+r^2}\right)\left(1+\frac{\left(-M+r+\sqrt{M^2+r^2}\right)^2}{2Mr}\right)}},\ \sqrt{\frac{r}{\left(M+\sqrt{M^2+r^2}\right)\left(1+\frac{\left(M-r+\sqrt{M^2+r^2}\right)^2}{2Mr}\right)}},\ 0,\ 0\right\},$$
$$\left.\left\{0,\ 0,\ \frac{1}{r},\ 0\right\},\ \left\{0,\ 0,\ 0,\ \frac{\text{Csc}[\theta]}{r}\right\}\right\}$$

Now the unmixed orthonormal basis vectors correspond.

```
In[77]:= ed[red@α] == Lud[a, red@α] ed[a]
         % // ToArrayValues[] // UseCoordinates[varnames] // TableForm
```

$$\text{Out[77]}= e_\alpha == e_a\ L^a{}_\alpha$$

Out[78]//TableForm=

$$e_0 == -\frac{\sqrt{M+r+\sqrt{M^2+r^2}}}{\sqrt{2}\ (M^2+r^2)^{1/4}}\ e_0 + \sqrt{\frac{r}{\left(-M+\sqrt{M^2+r^2}\right)\left(1+\frac{\left(-M+r+\sqrt{M^2+r^2}\right)^2}{2Mr}\right)}}\ e_1$$

$$e_1 == \frac{\sqrt{M+r-\sqrt{M^2+r^2}}}{\sqrt{2}\ (M^2+r^2)^{1/4}}\ e_0 + \sqrt{\frac{r}{\left(M+\sqrt{M^2+r^2}\right)\left(1+\frac{\left(M-r+\sqrt{M^2+r^2}\right)^2}{2Mr}\right)}}\ e_1$$

$$e_2 == \frac{e_2}{r}$$

$$e_3 == \frac{\text{Csc}[\theta]\ e_3}{r}$$

Again, we can check that the new basis vectors are orthonormal, and that the new metric is the Minkowski metric. However the algebraic expressions are so complicated that *Mathematica* won't simplify them. Therefore we will check for specific values of M and r.

```
In[79]:= Print["Check the orthogonality of the red basis vectors"]
         Outer[ed[#1].ed[#2] &, red /@ BaseIndices, red /@ BaseIndices] // MatrixForm
         Print["Substitute in terms of the coordinate basis"]
         %% /. ed[red@α_].ed[red@β_] → (Lud[a, red@α] ed[a]).(Lud[b, red@β] ed[b]) //
          MatrixForm
         Print["Evaluate the dot products"]
         %% // EvaluateDotProducts[e, g, False]
         Print["Perform the summations. The expressions are too difficult for
            Mathematica to simplify so check for a specific case, ", {M → 1, r → 3}]
         (%% // EinsteinSum[] // UseCoordinates[varnames]) /. {M → 1, r → 3} // MatrixForm
         % // FullSimplify // MatrixForm
```

Check the orthogonality of the red basis vectors

Out[80]//MatrixForm=

$$\begin{pmatrix} e_0.e_0 & e_0.e_1 & e_0.e_2 & e_0.e_3 \\ e_1.e_0 & e_1.e_1 & e_1.e_2 & e_1.e_3 \\ e_2.e_0 & e_2.e_1 & e_2.e_2 & e_2.e_3 \\ e_3.e_0 & e_3.e_1 & e_3.e_2 & e_3.e_3 \end{pmatrix}$$

Substitute in terms of the coordinate basis

Out[82]//MatrixForm=

$$\begin{pmatrix} (e_a L^a{}_0).(e_b L^b{}_0) & (e_a L^a{}_0).(e_b L^b{}_1) & (e_a L^a{}_0).(e_b L^b{}_2) & (e_a L^a{}_0).(e_b L^b{}_3) \\ (e_a L^a{}_1).(e_b L^b{}_0) & (e_a L^a{}_1).(e_b L^b{}_1) & (e_a L^a{}_1).(e_b L^b{}_2) & (e_a L^a{}_1).(e_b L^b{}_3) \\ (e_a L^a{}_2).(e_b L^b{}_0) & (e_a L^a{}_2).(e_b L^b{}_1) & (e_a L^a{}_2).(e_b L^b{}_2) & (e_a L^a{}_2).(e_b L^b{}_3) \\ (e_a L^a{}_3).(e_b L^b{}_0) & (e_a L^a{}_3).(e_b L^b{}_1) & (e_a L^a{}_3).(e_b L^b{}_2) & (e_a L^a{}_3).(e_b L^b{}_3) \end{pmatrix}$$

Evaluate the dot products

Out[84]=

$$\{\{g_{ab} L^a{}_0 L^b{}_0, g_{ab} L^a{}_0 L^b{}_1, g_{ab} L^a{}_0 L^b{}_2, g_{ab} L^a{}_0 L^b{}_3\},$$
$$\{g_{ab} L^a{}_1 L^b{}_0, g_{ab} L^a{}_1 L^b{}_1, g_{ab} L^a{}_1 L^b{}_2, g_{ab} L^a{}_1 L^b{}_3\},$$
$$\{g_{ab} L^a{}_2 L^b{}_0, g_{ab} L^a{}_2 L^b{}_1, g_{ab} L^a{}_2 L^b{}_2, g_{ab} L^a{}_2 L^b{}_3\},$$
$$\{g_{ab} L^a{}_3 L^b{}_0, g_{ab} L^a{}_3 L^b{}_1, g_{ab} L^a{}_3 L^b{}_2, g_{ab} L^a{}_3 L^b{}_3\}\}$$

Perform the summations. The expressions are too difficult for
Mathematica to simplify so check for a specific case, {M → 1, r → 3}

Out[86]//MatrixForm=

$$\begin{pmatrix} \frac{4+\sqrt{10}}{6\sqrt{10}} + \frac{2^{3/4}\sqrt{\frac{4+\sqrt{10}}{\left(-1+\sqrt{10}\right)\left(1+\frac{1}{6}\left(2+\sqrt{10}\right)^2\right)}}}{5^{1/4}} - \frac{3}{\left(-1+\sqrt{10}\right)\left(1+\frac{1}{6}\left(2+\sqrt{10}\right)^2\right)} \\ -\frac{1}{6}\sqrt{\frac{1}{10}\left(4-\sqrt{10}\right)\left(4+\sqrt{10}\right)} + \frac{\sqrt{\frac{4+\sqrt{10}}{\left(1+\sqrt{10}\right)\left(1+\frac{1}{6}\left(-2+\sqrt{10}\right)^2\right)}}}{10^{1/4}} - \frac{\sqrt{\frac{4-\sqrt{10}}{\left(-1+\sqrt{10}\right)\left(1+\frac{1}{6}\left(2+\sqrt{10}\right)^2\right)}}}{10^{1/4}} - \frac{}{\sqrt{\left(-1+\sqrt{10}\right)\left(1+\sqrt{10}\right)\left(1\right.}} \\ 0 \\ 0 \end{pmatrix}$$

Out[87]//MatrixForm=

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & -1 \end{pmatrix}$$

The new basis vectors are orthonormal for at least those values and give us the Minkowski metric matrix.

Restore state

```
In[88]:= DeclareBaseIndices @@ oldindices
         DeclareIndexFlavor @@ oldflavors;

In[90]:= Clear[Lmatrix, metric, cmetric, varnames, oldindices, oldflavors]
         ClearTensorValues[{gdd[i, j], guu[i, j]}]

In[92]:= ClearTensorShortcuts[{{x, e}, 1}, {{g, δ, G, L}, 2}, {Γ, 3}, {R, 4}]
```