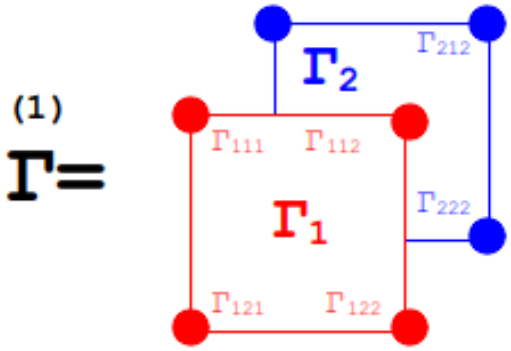
Exploring Math with MAXIMA

Tensor Calculus

A gentle Introduction

using the tensor packages of MAXIMA^{online}



Dr. Wolfgang Lindner
dr.w.g.Lindner@gmail.com
Leichlingen, Germany
version 2026

Contents

Preface

This booklet would like to whet your appetite to immerse yourself into the world of tensors with small bites of special tensors to take a closer look at tensor calculus. For the novice, tensor notations and operations are somehow clumsy and uncomfortable and accompanied by heavy calculations. Therefore here is no forbidding theory presented but instead the focus is on praxis with selected examples - using MAXIMA as your computer algebraic companion to unburden calculation in this field. We only cite some of the necessary underlying mathematical definitions and facts to be able to show the corresponding implementation of the concepts into the language of CAS MAXIMA and the special packages `i/ctensor`.

For the theory of Tensor Analysis I recommend the treatments by Pavel GRINFELD and Peter COLLIER. Their books [7] resp. [5] focusses on comprehension and illuminates mathematical concepts by well choosen worked examples and exercises. Daniel FLEISCH also offers a very understandable introduction in [6]. LEVITT [1, Appendix.1] offers a very short (25 pp.) and precise introduction to Tensors.
Looking back at my first contact with MAXIMA `ctensor` package I was very impressed by the ingenious and sophisticated treatment of the RIEMANN tensor in the calculation of the Schwarzschild metric by Viktor TOTH in the suite of examples in [39]. This example's potpourri is an eye opener and presents convincing powerful examples of the amazing efficiency of MAXIMA's tensor packages.

The collection of 24 short MAXIMA example scripts and 35 exercises in this booklet to cope with tensor operations not only want to help the reader to dive into the praxis of tensor calculations, but also to become comfortable with the use of the CAS MAXIMA in this field. It aims to be a source of help and inspiration for another round of thoughtful activity with respect to tensors - supplementary to paper & pencil calculations, now from the perspective of a compact CAS. So have fun on our tour from the omnipresent metric tensor via KRONECKER delta to the CHRISTOFFEL symbol. My small booklet on Elementary Differential Geometry [11] will e.g. address the RIEMANN tensor R in $\mathbb{R}^3$.
For the inspection or running an MAXIMA script no installation is necessary, *everything runs directly online*: a click on a link like ▷ Click here to RUN the code. in this text is enough to invoke the corresponding script and execute it automatically.
The code lines were tested on an iMac 24", Apple M3, Tahoe 26.3.1 using Safari.

I want to thank Viktor TOTH and Michel GOSSE for their friendly support with tips and hints while writing these notes. Without Viktor's expert help and contributions it would not have been possible for me to present the/his smart examples using the `ctensor` package.

Wolfgang Lindner
Leichlingen, Germany, May 2026
Comments? Suggestions? Critique? Email the author at dr.w.g.Lindner@gmail.com

1 Coordinate Systems and Transformations

The description of tensors relies largely on transformations between coordinate systems. Therefore the reader should be familiar with the basics of linear algebra (vector spaces, transformations) and analysis (partial derivative). FLEISCH [6] offers a very understandable introduction. We begin with some examples of coordinate systems and the switch between different basis with transformations (linear mappings), and also introduce the terminology and notations used in this book.

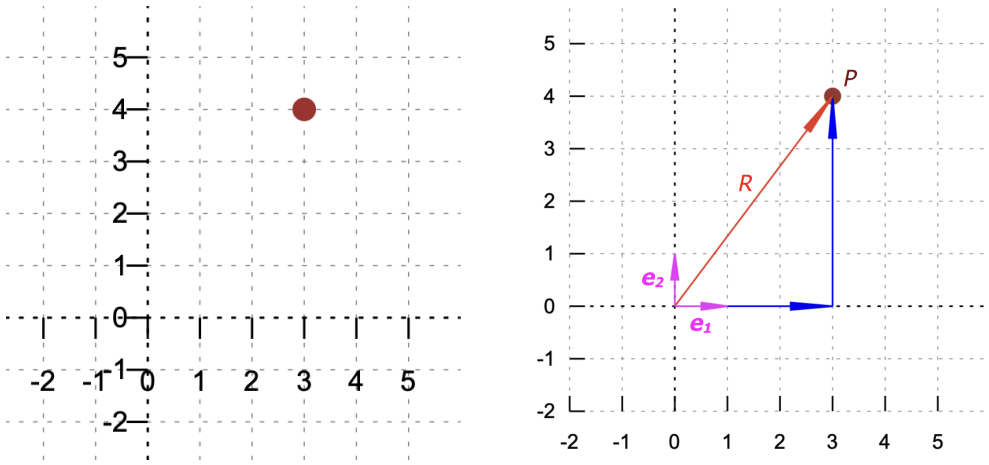


Figure 1: left: point P(3,4) with coordinates $x = 3$ and $y = 4$ in $\mathbb{R}^2$.
right: the standard basis $\mathbf{e}_1 = \begin{bmatrix}1\\0\end{bmatrix}$, $\mathbf{e}_2 = \begin{bmatrix}0\\1\end{bmatrix}$, the position vector $R = \begin{bmatrix}3\\4\end{bmatrix}$ and its decomposition '$\rightarrow\uparrow$' of R into $3\cdot\ \mathbf{e}_1 + 4\cdot\ \mathbf{e}_2$

In general, we associate to a point $P \in \mathbb{R}^n$ an n-tuple $(x^1_P, x^2_P, .., x^n_P)$, where *the superscripts does not denote the powers* of the numbers, but the i-th entry in the given coordinate system tuple. As we will see, this habit is in harmony with the usual tensor notation.

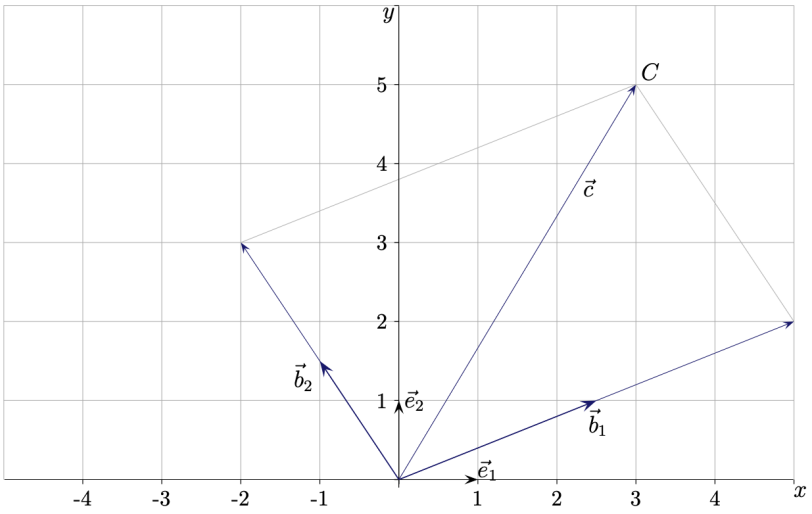
◦ For example, EINSTEIN wrote the usual sum $a_1e_1 + a_2e_2 + a_3e_3$ with superscript indices as $a^1e_1 + a^2e_2 + a^3e_3$ and abbreviated this summation to a^ie_i, where summation is to be performed over indices with the same name above and below.
◦ Or: $b^1e_1 + b^2e_2 \equiv b^ie_i$.
◦ In general we have: $a^1e_1 + a^2e_2 + ... + a^ne_n = \sum_{k=1}^n a^ke_k =: a^ke_k$.
We see, that this notation is independent of the dimension of the space $\mathbb{R}^n$.
This abbreviation is called the

> (EINSTEIN summation convention)*: A summation in a tensor expression is implicit done, when an index appears twice, once as a subscript ('lower index') and once as a superscript ('upper index').*

• If one writes $A_i{}^k \cdot B_k{}^j$, this is *interpreted in tensor language* as *a hidden summation behind the scene*, e.g. we suppress the summation sign $\sum_{k=1}^n$ in such tensor expressions.

1.1 Examples

Example 1. (Same point, but different coordinates w.r.t. a basis)
We look at two coordinate systems in the 2D Euclidean vectorspace $\mathbb{R}^2$ with canonical scalar product $\bullet$, i.e. we have two different basis vectors $\vec{e}_1, \vec{e}_2$ resp. $\vec{b}_1, \vec{b}_2$ written in the physicist notation with an overline arrow $\vec{..}$.
Let by example $\vec{e}_1 = \begin{bmatrix}1\\0\end{bmatrix}$, $\vec{e}_2 := \begin{bmatrix}0\\1\end{bmatrix}$ and $\vec{b}_1 := \begin{bmatrix}2.5\\1\end{bmatrix}$, $\vec{b}_2 := \begin{bmatrix}-2\\3\end{bmatrix}$.
We ask: what are the coordinates of point $C(3,5)_e$ with it's coordinates w.r.t. basis $e = (e_1, e_2)$ in the other coordinate system $b = (b_1, b_2)$, i.e $C(?,?)_b$?



$\vec{e}_1, \vec{e}_2$: the standard Euclidean basis in $\mathbb{R}^2$.
Figure 2: $\vec{b}_1, \vec{b}_2$: another nonstandard basis in $\mathbb{R}^2$.
C: a point in $\mathbb{R}^2$ with different coordinates w.r.t. e resp. b.

First we drop the physicist notation in fig.1[1] and write vectors in **boldface** or in CAS MAXIMA[2] using capital letters, i.e. we write $\vec{e}_1$ as $\mathbf{e}_1$ or $E1$.

1st we solve the problem using elementary (linear) algebra.
We express the point C more precise and exact by its development in the basis vectors (see fig.1) and then do the following steps:

1. $C = (3,5) \equiv 3 \cdot E_1 + 5 \cdot E_2 = u \cdot B_1 + v \cdot B_2$ unknown coord's u, v of C w.r.t basis b
2. $B_1 = 2.5 \cdot E_1 + 1 \cdot E_2$ express B_1 w.r.t. basis e
   $B_2 = -1 \cdot E_1 + 1.5 \cdot E_2$ express B_2 w.r.t. basis e
3. $3 \cdot E_1 + 5 \cdot E_2 = u \cdot (2.5 \cdot E_1 + 1 \cdot E_2) + v \cdot (-1 \cdot E_1 + 1.5 \cdot E_2)$ substitute 2. in 1.

[1] The figure is found in ROOLFS, [23].
[2] which don't have boldface or overline arrows

4. $3 \cdot E_1 + 5 \cdot E_2 = 2.5u \cdot E_1 + u \cdot E_2 + (-1)v \cdot E_1 + 1.5v \cdot E_2$ evaluate 3.

5. $3 \cdot E_1 + 5 \cdot E_2 = (2.5u - v) \cdot E_1 + (1u + 1.5v) \cdot E_2$ collect

6. $(3, 5) = (2.5u - v, 1u + 1.5v)$ write component wise

7. $\begin{bmatrix} 3 \\ 5 \end{bmatrix} = \begin{bmatrix} 2.5 & -1 \\ 1 & 1.5 \end{bmatrix} \bullet \begin{bmatrix} u \\ v \end{bmatrix}$ write as matrix equation

8. $\begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} \frac{6}{19} & \frac{4}{19} \\ \frac{-4}{19} & \frac{10}{19} \end{bmatrix}^{-1} \bullet \begin{bmatrix} 3 \\ 5 \end{bmatrix}$ use inverse matrix (check!)

9. $\begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} 2 \\ 2 \end{bmatrix}$ read off the solution

Result: the coordinates of point $C(3, 5)_e$ in the other coordinate system $b = (b_1, b_2)$ are $C(2, 2)_b$. This is verified by looking in fig.2.

2nd we solve the problem using MAXIMA following the above 9 steps.

```
e1 : [1, 0];
e2 : [0, 1];
b1 : [2.5, 1];
b2 : [-1, 1.5];

C : [3,5];     /* w.r.t. base e, i.e.: */

3*e1 + 5*e2 = u*b1 + v*b2;

[3,5] = [2.5*u-v,1.5*v+u];

/* we solve for u,v using function linsolve() */
linsolve ([3 = 5/2*u - v, 5 = 3/2*v + u], [u,v]);

/* alternative we solve using inverse matrix */
A : matrix([5/2,  -1],  [1, 3/2]);
A^^(-1);
A^^(-1) . C;
```

▷ Click here to RUN the code.

MAXIMA output:

```
[%i10] A : matrix([5/2,  -1],  [1, 3/2]);
```

[%o10]
$$\begin{pmatrix} \frac{5}{2} & -1 \\ 1 & \frac{3}{2} \end{pmatrix}$$

```
[%i11] A^^(-1);
```

[%o11]
$$\begin{pmatrix} \frac{6}{19} & \frac{4}{19} \\ -\frac{4}{19} & \frac{10}{19} \end{pmatrix}$$

```
[%i12] A^^(-1) . C;
```

[%o12]
$$\begin{pmatrix} 2 \\ 2 \end{pmatrix}$$

3rd we generalize the problem using general values for point C and basis b.

```
e1 : [1, 0];
e2 : [0, 1];
C  : [x, y];                                  /* 1 */

/* other basis vectors b1, b2 */
b1 : a11*e1 + a21*e2;
b2 : a12*e1 + a22*e2;

/* ansatz for same point C(x,y) w.r.t. e and b */
x*e1 + y*e2 = u*b1 + v*b2;                    /* 2 */

/*  read coordinates of C w.r.t. basis b */
x : a11*u + a12*v;
y : a21*u + a22*v;

/* collect new coordinates of C as columns(!) in a matrix A */
/* i.e. A is transpose of (b1,b2) */
A : matrix([a11,  a12], [ a21, a22]);

kill(x,y)$

[x,y] = A . [u,v];                            /* 3 */

/* X = A . U => U = inverse(A) . X */
transpose([u,v]) = A^^(-1) . [x,y];           /* 4 */
```

▷ Click here to RUN the code.

Remark. In 1: we represent the point C as a list (array) in MAXIMA. One should correctly represent it as a column vector `C : matrix([x],[y])` - but this looks a bit costly at the

moment. In 2: you read the coordinates x and y of C w.r.t. e now expressed via the new coordinates (u, v) w.r.t. b. Line 3: writes 2: in matrix form, so that we can solve afterwards for the new coordinates (u, v) with matrix methods (inverse matrix). 4: use transposed (u, v), so that we see both arranged in a column for better read the solution on the right side. Surely: one should better write `matrix([x],[y])` or `transpose([x,y])` on the right side of 4: ... like (i12).
Nevertheless: you read off the solution formula for the new coordinates of C.

MAXIMA output:

```
[%i11] [x,y] = A . [u,v];
```

$$[\%o11]\quad [x,y] = \begin{pmatrix} a_{12}\,v + a_{11}\,u \\ a_{22}\,v + a_{21}\,u \end{pmatrix}$$

```
[%i12] /* X = A . U => U = inverse(A) . X */
       transpose([u,v]) = A^^(-1) . [x,y];
```

$$[\%o12]\quad \begin{pmatrix} u \\ v \end{pmatrix} = \begin{pmatrix} \dfrac{a_{22}\,x}{a_{11}\,a_{22} - a_{12}\,a_{21}} - \dfrac{a_{12}\,y}{a_{11}\,a_{22} - a_{12}\,a_{21}} \\ \dfrac{a_{11}\,y}{a_{11}\,a_{22} - a_{12}\,a_{21}} - \dfrac{a_{21}\,x}{a_{11}\,a_{22} - a_{12}\,a_{21}} \end{pmatrix}$$

Exercise 1. Use the general formula (o12) to redo the calculation in 1^{st} or 2^{nd} in one step.

Exercise 2. Calculate the length of C in the selected basis e resp. b, i.e.
determine $|C_e|$ and $|C_b|$, where $|X| := \sqrt{X \bullet X}, X \in \mathbb{R}^2$.

Exercise 3. Calculate the angle between $\vec{OC}$ and the first 'axis' e_1 resp. b_1, i.e. determine:
$\angle(C_e, e_1)$ and $\angle(C_b, b_1)$.

Example 2. (Coordinate change by rotation)
We look at a counterclockwise rotation e.g. by 30° of the standard basis in the 2D Euclidean vectorspace $\mathbb{R}^2$. We ask: how does the coordinates of a point P change? What is the transformation rule?
Let by example $\vec{e}_1 := \begin{bmatrix}1\\0\end{bmatrix}$, $\vec{e}_2 := \begin{bmatrix}0\\1\end{bmatrix}$ and $\vec{e}_1^{\,*} := T_{30^\circ}(\vec{e}_1)$, $\vec{e}_2^{\,*} := T_{30^\circ}(\vec{e}_2)$, where T_α represents the counterclockwise rotation with an angle of α.

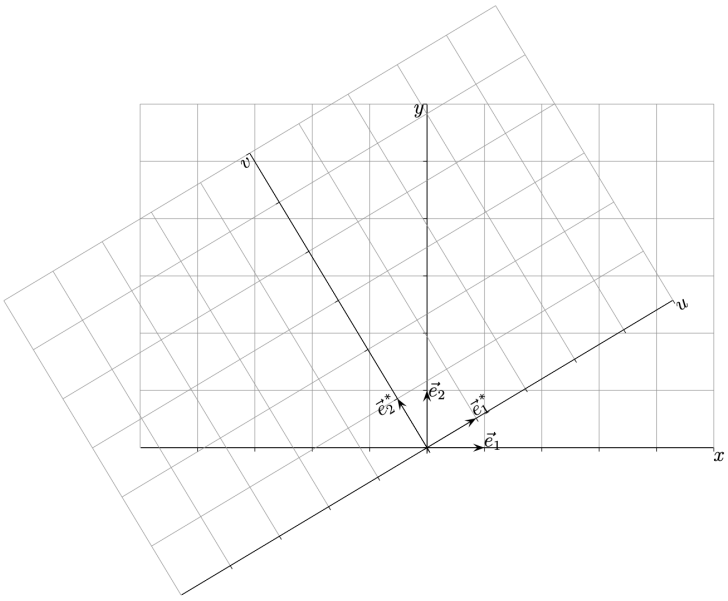


Figure 3: $\vec{e}_1, \vec{e}_2$: the standard Euclidean basis in $\mathbb{R}^2$.
$\vec{e}_1^{\,*}, \vec{e}_2^{\,*}$: the rotated standard basis. [Picture by ROOLFS].

Determine

a. the matrix of the transformation T_α, the counterclockwise rotation with an angle of α and T_{30°.

b. the coordinates of point $P(2,3)_e$ with it's coordinates w.r.t. basis $\vec{e} = (\vec{e}_1, \vec{e}_2)$ in the rotated coordinate system $\vec{e}^{\,*} = (\vec{e}_1^{\,*}, \vec{e}_2^{\,*})$, i.e $P^*(?,?)$?

c. Be warned: *Not every matrix represents a tensor.*

Show: Let $A = \begin{bmatrix}a_{11} & a_{21}\\ a_{12} & a_{22}\end{bmatrix}$ be orthogonal[3] and $T = \begin{bmatrix}T^{11} & T^{21}\\ T^{12} & T^{22}\end{bmatrix}$.

If $T^* = A^t \bullet T \bullet A$, then T represents a tensor
and in EINSTEIN's tensor notation we have $T^*_{ij} = a_{ki} a_{lj} T^{kl}$

[3] i.e. $A^{-1} = A^t$, which is the case for the transformation T_α from part a., where A is the invertible matrix of a coordinate change.

Solution:

Let's drop again the physicist notation and write vectors simplified noted as $\mathbf{e}_1$ or $E1$ etc.

1st we solve problems a. and b. using elementary Linear Algebra.

1. $T_\alpha = \begin{bmatrix}\cos(\alpha) & -\sin(\alpha)\\ \sin(\alpha) & \cos(\alpha)\end{bmatrix}$ known from Linear Algebra

2. $T_{30^\circ} = \begin{bmatrix}\cos(\pi/6) & -\sin(\pi/6)\\ \sin(\pi/6) & \cos(\pi/6)\end{bmatrix} = \begin{bmatrix}\frac{\sqrt{3}}{2} & -\frac{1}{2}\\ \frac{1}{2} & \frac{\sqrt{3}}{2}\end{bmatrix}$

3. $P = (2,3) \equiv 2 \cdot E_1 + 3 \cdot E_2 = u \cdot E_1^* + v \cdot E_2^*$ unknown u, v of P w.r.t. basis e^*

   general approach for transformation from E to E^*:

4. $E_1^* = a_{11} \cdot E_1 + a_{12} \cdot E_2$ express e_1^* w.r.t. basis e
   $E_1^* = a_{21} \cdot E_1 + a_{22} \cdot E_2$ express e_2^* w.r.t. basis e

   From 1. we have $a_{11} = \cos(\alpha), a_{12} = \sin(\alpha)$, etc., ergo

5. $\begin{bmatrix}x\\ y\end{bmatrix} = T_\alpha(\begin{bmatrix}u\\ v\end{bmatrix}) = \begin{bmatrix}\cos(\alpha) & -\sin(\alpha)\\ \sin(\alpha) & \cos(\alpha)\end{bmatrix} \cdot \begin{bmatrix}u\\ v\end{bmatrix}$ transformation rule $(u,v) \mapsto (x,y)$

6. $\begin{bmatrix}u\\ v\end{bmatrix} = T_\alpha^{-1}(\begin{bmatrix}x\\ y\end{bmatrix}) = \begin{bmatrix}\cos(\alpha) & \sin(\alpha)\\ -\sin(\alpha) & \cos(\alpha)\end{bmatrix} \cdot \begin{bmatrix}x\\ y\end{bmatrix}$ inverse transformation $(x,y) \mapsto (u,v)$

7. $\begin{bmatrix}u\\ v\end{bmatrix} = \begin{bmatrix}\frac{\sqrt{3}}{2} & -\frac{1}{2}\\ \frac{1}{2} & \frac{\sqrt{3}}{2}\end{bmatrix}^{-1} \cdot \begin{bmatrix}2\\ 3\end{bmatrix}$

8. $\begin{bmatrix}u\\ v\end{bmatrix} \approx \begin{bmatrix}3.2\\ 1.6\end{bmatrix}$ read the solution

Result: the coordinates of point $P(2,3)_e = 2 \cdot \vec{e}_1 + 3 \cdot \vec{e}_2$ in the other coordinate system e^* are $P^*(3.2, 1.6)$, i.e. $P^* = 3.2 \cdot \vec{e}_1^* + 1.6 \cdot \vec{e}_2^*$. This is verified by looking in fig.3.

2nd we solve problems a. and b. using MAXIMA .

```
T(alpha) := matrix([cos(alpha),-sin(alpha)], [sin(alpha), cos(alpha)]);
T(%pi/6);
invert(T(alpha));
Tinv : trigsimp(invert(T(alpha)));
Tinv : trigsimp(invert(T(%pi/6)));
Tinv . [2,3];
Tinv . [2,3], numer;
```

▷ Click here to RUN the code. MAXIMA output:

```
[%i4] Tinv : trigsimp(invert(T(alpha)));
```

[%o4] $$\begin{pmatrix} \cos\alpha & \sin\alpha \\ -\sin\alpha & \cos\alpha \end{pmatrix}$$

```
[%i5]
```

[%o5] $$\begin{pmatrix} \frac{\sqrt{3}}{2} & \frac{1}{2} \\ -\frac{1}{2} & \frac{\sqrt{3}}{2} \end{pmatrix}$$

```
[%i6] Tinv . [2,3];
```

[%o6] $$\begin{pmatrix} \sqrt{3}+\frac{3}{2} \\ \frac{3\frac{3}{2}}{2}-1 \end{pmatrix}$$

```
[%i7] Tinv . [2,3], numer;
```

[%o7] $$\begin{pmatrix} 3.232050807568877 \\ 1.598076211353316 \end{pmatrix}$$

Exercise 4. Verify, that the length of P is independent of the selected basis e resp. e^*, i.e. show: $|P| = |P^*|$, where $|X| := \sqrt{X \bullet X}, X \in \mathbb{R}^2$. Control in the sketch with a ruler.

Exercise 5. Verify, that the angle between $\vec{OP}$ and the first 'axis' $E1$ resp. E_1^* is independent of the selected basis e resp. e^*, i.e. show: $\angle(P, E1) = \angle(P^*, E1^*)$. Control in the sketch with the protractor.

3rd we solve c. using MAXIMA.

```
A : matrix([a11, a12], [a21, a22]);
/* A^-1 = A^t, assumed A orthogonal! */
At : transpose(A);
T : matrix([T11,T12], [T21, T22]);
(At . T . A);
```

▷ Click here to RUN the code.
MAXIMA output:

```
[%i1] A : matrix([a11, a12], [a21, a22]);
```

[%o1] $$\begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix}$$

```
[%i2] /* A^-1 = A^t, A orthogonal! */
      At : transpose(A);
```

[%o2] $$\begin{pmatrix} a_{11} & a_{21} \\ a_{12} & a_{22} \end{pmatrix}$$

```
[%i3] T : matrix([T11,T12], [T21, T22]);
```

[%o3] $$\begin{pmatrix} T_{11} & T_{12} \\ T_{21} & T_{22} \end{pmatrix}$$

```
[%i4] (At . T . A), expand;
```

[%o4] $$\begin{pmatrix} T_{22}\,a_{21}{}^2 + T_{21}\,a_{11}\,a_{21} + T_{12}\,a_{11}\,a_{21} + T_{11}\,a_{11}{}^2 & T_{22}\,a_{21}\,a_{22} + T_{12}\,a_{11}\,a_{22} + T_{21}\,a_{12}\,a_{21} + T_{11}\,a_{11}\,a_{12} \\ T_{22}\,a_{21}\,a_{22} + T_{21}\,a_{11}\,a_{22} + T_{12}\,a_{12}\,a_{21} + T_{11}\,a_{11}\,a_{12} & T_{22}\,a_{22}{}^2 + T_{21}\,a_{12}\,a_{22} + T_{12}\,a_{12}\,a_{22} + T_{11}\,a_{12}{}^2 \end{pmatrix}$$

Now you may gather from (%o4) the RHS of the EINSTEIN notation $T^*_{ij} = a_{ki}a_{lj}T^{kl}$.

Comment. We elaborate a little on (%i4). Suppose we have the coordinates $U = [u, v]^t$ in basis e^* and the corresponding coordinates $X = [x, y]^t$ w.r.t. basis e. Then (%i4) reflects the coordinate change $T^* \cdot U = (A^t \cdot T \cdot A) \cdot U$, which is the composite mapping $U \xleftarrow{A^t} X^* \xleftarrow{T} X \xleftarrow{A} U$ and shows, that T leaves the coordinates invariant.

Example 3. (Inertia Tensor)
We follow ROOLFS [22]. We consider a system of rigidly coupled point masses of identical mass $m = 1$.

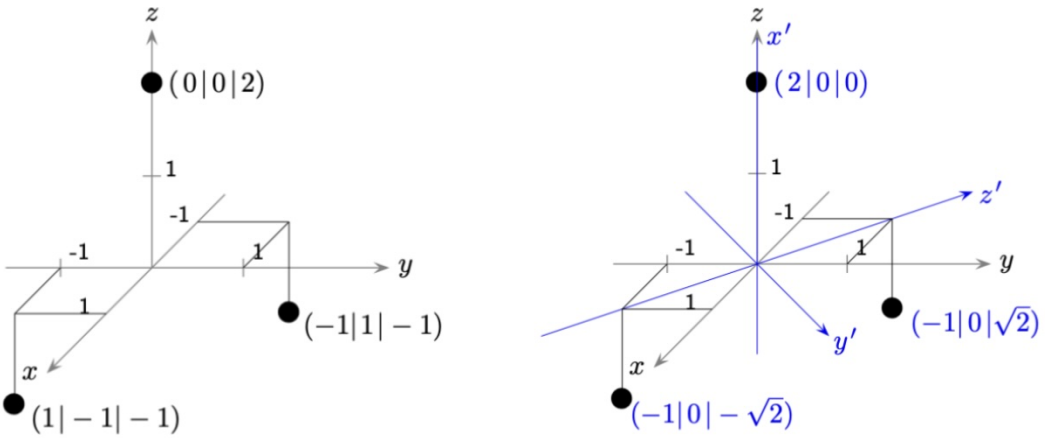


Figure 4: •left: three coupled point masses in $\mathbb{R}^3$ w.r.t. standard basis.
•right: the points coordinates in the basis of the eigenvectors.

Verify the following facts about the configuration of fig.2:

a. Verify, that the inertia tensor Θ of the configuration in fig.2^{left} can be determined as

$$\Theta = \begin{bmatrix} 8 & 2 & 0 \\ 2 & 8 & 0 \\ 0 & 0 & 4 \end{bmatrix}$$

w.r.t. the Euclidian basis $e_1 = (1, 0, 0)^t, e_2 = (0, 1, 0)^t, e_3 = (0, 0, 1)^t$ of $\mathbb{R}^3$, see [22].

b. Verify, that the eigenvalues of matrix Θ are $\lambda_1 = 4, \lambda_2 = 10, \lambda_3 = 6$ with corresponding eigenvectors $w_1 = (1, -1, 0)^t, w_2 = (1, 1, 0)^t, w_3 = (0, 0, 1)^t$.
Calculate the normalized eigenvectors (v_1, v_2, v_3) from (w_1, w_2, w_3) via $v_i := w_i/|w_i|$.

c. Use the matrix $A := [v_1, v_2, v_3]$ of the normalized eigenvectors[4] to calculate the new coordinates of the mass points w.r.t. the basis $v = (v_1, v_2, v_3)$. Check your result in fig.$4.^{right}$. [Tip: remember step 8. resp. formulas *3* and *4* of example.1.]

[4] the normalized eigenvectors v_i are the *columns* of matrix A.

d. Verify, that the matrix Θ is a *tensor*, i.e. Θ transforms via

$$\Theta' = \begin{bmatrix} 4 & 0 & 0 \\ 0 & 10 & 0 \\ 0 & 0 & 6 \end{bmatrix} = A^t \cdot \Theta \cdot A$$

To quote ROOLFS:

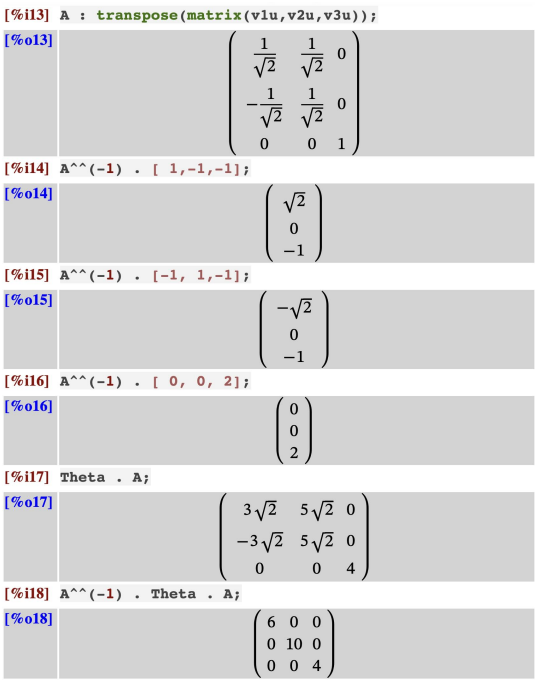
> "The diagonalization is not the remarkable aspect, but rather that $\Theta' = A^t \cdot \Theta \cdot A$ correctly describes the rotation around the new coordinate system. Therefore, it is not simply a renaming of the coordinates."

1st We (re)do the calculations a. ... d. with MAXIMA.

```
Theta : matrix([8,2,0],[2,8,0],[0,0,4]);
ev : eigenvalues (Theta );
Ev : eigenvectors(Theta );

/* b:  the eigenvectors of Theta */
v1 : [1, -1,  0];
v2 : [1,  1,  0];
v3 : [0,  0,  1];

/* Check if Theta . v_i = lambda_i * v_i */
Theta . v1;
Theta . v2;
Theta . v3;

v1u : unitvector (v1);
v2u : unitvector (v2);
v3u : unitvector (v3);

/* c:  A is matrix of normed eigenvectors as columns */
A : transpose(matrix(v1u,v2u,v3u));

/*     calculate the new coordinates U via */
/*     X = A . U => U = inverse(A) . X */

A^^(-1) . [ 1,-1,-1];
A^^(-1) . [-1, 1,-1];
A^^(-1) . [ 0, 0, 2];

/* d:  verify tensor property of Theta */
Theta . A;
A^^(-1) . Theta . A;
```

▷ Click here to RUN the code.

MAXIMA output:



2nd We now write the tensor transformation formula $\Theta' = A^t \cdot \Theta \cdot A$ using index notation.

Suppose we have two matrices $A = (A_{ij})_{i=1..m}^{j=1..n}$ and $B = (B_{jk})_{j=1..n}^{k=1..p}$, then the $_{ik}$'s element C_{ik} of the product of two matrices A and B is calculated/defined as

$$\begin{aligned} C_{ik} &= A_{i,1} \cdot B_{1,k} + A_{i,2} \cdot B_{2,k} + \cdots + A_{i,m} \cdot B_{m,k} \\ &= \sum_{k=1}^{n} A_{ik} \cdot B_{kj} \\ &=: A_i{}^k \cdot B_k{}^j \qquad \leftarrow \text{EINSTEIN } \textit{summation convention} \end{aligned}$$

Remark: In the last expression we have discriminated precise between the *row indices* and the *column indices* of a matrix using a subscript ('lower index') for the row indices and a superscript ('upper index') for the column indices. Then we can apply the EINSTEIN summation convention, see section 1.0.

- Ergo, summation symbols $\sum$ can be 'eliminated or forgotten' by using EINSTEIN's convention, giving very compact tensorial notations - but with the drawback to decode the terms in your mind.

If we have a third matrix C with $C = (C_{k\ell})_{k=1..p}^{\ell=1..q}$, then the triple product $A \cdot B \cdot C$[5] is calculated via

[5]Remember: the matrix product is associative.

$$\begin{aligned}((A\cdot B)\cdot C)_{i=1..m}^{\ell=1..q} &= \sum_{k=1}^{p}(A\cdot B)_{ik}\cdot C_{k\ell} = \sum_{k=1}^{p}(\sum_{j=1}^{n}A_{ij}\cdot B_{jk})\cdot C_{k\ell}\\ &= \sum_{k=1}^{p}(\sum_{j=1}^{n}A_i^j\cdot B_j^k)\cdot C_k^\ell \overset{\text{EINSTEIN}}{=} A_i^j\cdot B_j^k\cdot C_k^\ell\end{aligned}$$

We now specialize this to our matrices Θ and A and conclude

$$\begin{aligned}(\Theta')_{i=1..3}^{\ell=1..3} &= \sum_{k=1}^{3}(\sum_{j=1}^{3}(A^t)_i^j\cdot\Theta_j^k)\cdot A_k^\ell\\ &= \sum_{k=1}^{3}\sum_{j=1}^{3}((A^t)_i^j\cdot\Theta_j^k)\cdot A_k^\ell\\ &= (A^t)_i{}^j\cdot\Theta_j{}^k\cdot A_k{}^\ell\end{aligned}$$

Therefore we get the tensor formula for the transformation rule

$$(\Theta')_i^\ell = (A^t)_i{}^j\cdot\Theta_j{}^k\cdot A_k{}^\ell$$

with A as invertible transformation, i.e. $A^t = A^{-1}$.

- Θ is a tensor of type $(1,1)$, because we have one subscript and one superscript index.

3rd We formulate the tensor transformation formula $\Theta' = A^t\cdot\Theta\cdot A$ using index notation and the tensor package `itensor`.

```
load("itensor")$
ishow( At([i],[j]) * Theta([j],[k]) * A([k],[l]) )$
indices(%);
```

We see: in the term `A([k],[l])` the first bracket [ ] collects the *sub*scripted indices and the second [ ] the *super*script indices. MAXIMA output:

```
[%i1] load("itensor")$
[%i2]
[%o2]   j  k  l
      At  θ  A
        i  j  k
[%i3] indices(%);
[%o3] [[l, i], [j, k]]
```

▷ Click here to RUN the code.

2 What is a tangent vector?

We begin with tree examples of tangent vectors to gain a visual mental image of this concept, which plays a central role beside the position vectors. Imagine a tangent vector as a vector that is tangent at a given point to a curve, a surface or a higher dimensional space, i.e. a 'manifold'. *Tangent vectors are examples of tensors* of rank 1, type $(1,0)$.

Example 4. (Tangent vector on a Parabola)
Imagine a mass point that travels along a parabolic path over time at a certain speed. If the mass point leaves the path at a certain position, it leaves the parabolic path *tangentially*. We denote the path by M ('manifold'), the position vector of the point at time t by $R(t)$ and the velocity by v.

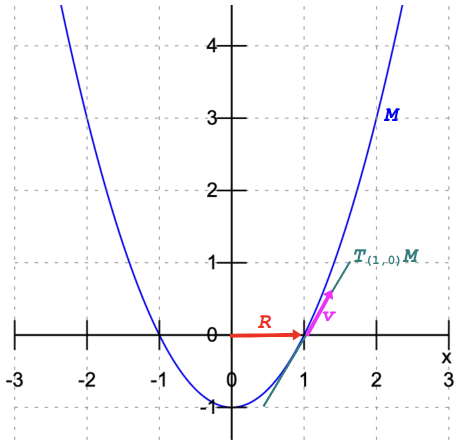


Figure 5: M: parabola $\{(x,y) \in \mathbb{R}^2 | y = x^2 - 1\}$.
R: the position vector $R(1) = (0,1)$.
v: the velocity vector $R'(t) = (1, 2t)$.
$T_{(0,1)}M$: tangent line at point $P = R(1)$ on M.

Then the velocity is defined as follows:

$$\mathbf{v} := \underbrace{\frac{\partial R}{\partial t}}_{\text{Math}} \qquad \underbrace{= \text{diff}(R(t), t)}_{\text{Maxima}}$$

• We calculate the velocity v in Fig.5. first mathematically.

Let $R : I \to \mathbb{R}^2$ with $t \mapsto (t, t^2)$ be differentiable on the open interval $I \subset \mathbb{R}$.
Then $R'(t) = \frac{\partial}{\partial t}(t, t^2 - 1) = (1, 2t)$ and $R'(1) = (1, 2) = v$ is the *tangential vector* at the time $t = 1$ alias at the point $P = (0, 1)$.

• Then we calculate v using MAXIMA:

```
R(t) := (t,t^2-1);          /* parametrization of parabola */
v : diff(R(t),t);           /* velocity  */
vP : subst(t=1, v);         /* velocity at t=1 */
Tp : [1,vP];                /* tangent vector at [0,1] */
```

▷ Click here to RUN the code.

Exercise 6. (tangent vector on a circle)
The unit circle is a 1-dimensional manifold ('curve') in $\mathbb{R}^2$.
We recommend to describe this circle $S^1 := \{(x, y) \in \mathbb{R}^2 | x^2+y^2 = 1\}$ via a parametrization using the position vector $R(t) := (\cos(t), \sin(t))$.
Calculate the tangent vector at $P = R(45°)$.

You should reproduce the following scene:

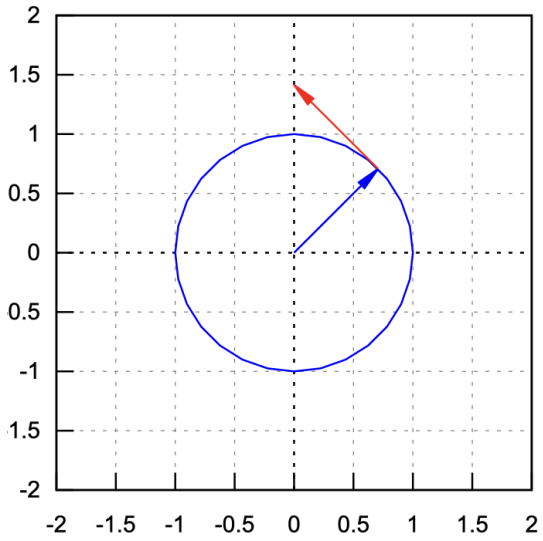


Figure 6: $R(\pi/4)$: position vector at time $t = \pi/4$ pointing on S^1.
$\partial_t R$: tangent vector $R'(\pi/4)$ at $R(\pi/4) \in S^1$.

Example 5. (Tangent vector on a Sphere)
The unit sphere S^2 is a 2-dimensional manifold ('surface') in the surrounding space $\mathbb{R}^3$.
Let's describe the sphere $S^2 := \{(x, y, z) \in \mathbb{R}^3 | x^2+y^2+z^2 = 1\}$ by means of a parametrization using the position vector $R(\theta, \phi)$.

Let $R : U \to \mathbb{R}^3$ be defined by

$$(\theta, \phi) \mapsto (\cos(\theta)\cos(\phi), \cos(\theta)\sin(\phi), \sin(\theta))$$

and differentiable on an open subset $U \subset \mathbb{R}^2$ with $(\theta, \phi) \in U$.

Then we have two tangent vectors: $\mathbf{Z_1} := \frac{\partial R}{\partial \theta}$ and $\mathbf{Z_2} := \frac{\partial R}{\partial \phi}$ are the *tangential vectors* at the point $R(\theta, \phi) \in S^2$ with coordinates (θ, ϕ), which span the tangential plane $T_P M$.

Task:
a. Calculate the tangent vectors at $P = R(\theta, \phi)$.
b. Calculate the tangent vectors at $P = R(45°, 0)$.

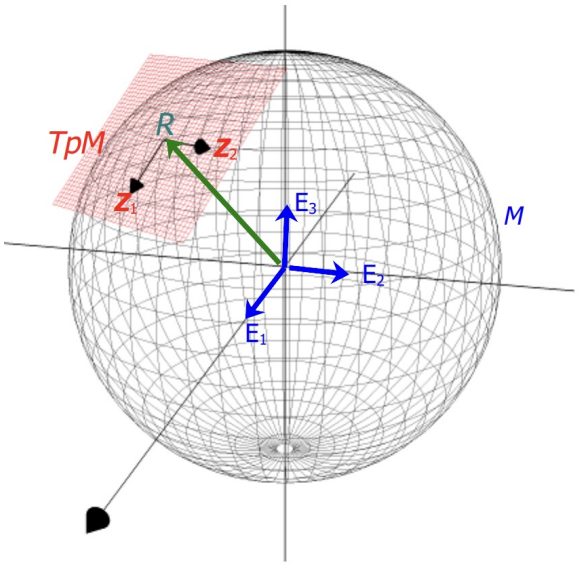


Figure 7: (E_1, E_2, E_3): canonical Euclidian basis in $\mathbb{R}^3$.
(Z_1, Z_2): basis of tangent plane $T_P S^2$ at $R(\theta, \phi) \in S^2$.
R: the position vector from the origin O to $R(\theta, \phi)$.

Solution:
$\mathbf{1}^{st}$ mathematically.
a:

$$\begin{aligned}
\mathbf{Z_1} &= \frac{\partial R}{\partial \theta} \\
&= \frac{\partial}{\partial \theta}(\cos(\theta)\cos(\phi), \cos(\theta)\sin(\phi), \sin(\theta)) \\
&= (-\cos(\phi)\sin(\theta), -\sin(\phi)\sin(\theta), \cos(\theta))
\end{aligned}$$

$$\begin{aligned}
\mathbf{Z_2} &= \frac{\partial R}{\partial \phi} \\
&= \frac{\partial}{\partial \phi}(\cos(\theta)\cos(\phi), \cos(\theta)\sin(\phi), \sin(\theta)) \\
&= (-\sin(\phi)\cos(\theta), \cos(\phi)\cos(\theta), 0))
\end{aligned}$$

b:

$\mathbf{Z_1} \mid (\frac{\pi}{4}, 0) = (\frac{-1}{\sqrt{2}}, 0, \frac{1}{\sqrt{2}})$
$\mathbf{Z_2} \mid (\frac{\pi}{4}, 0) = (0, \frac{1}{\sqrt{2}}, 0)$

$\mathbf{2}^{nd}$ solution with MAXIMA.

```
R(theta,phi) := [cos(theta)*cos(phi), cos(theta)*sin(phi),  sin(theta)]$

Z1 : diff( R(theta,phi), theta);
Z2 : diff( R(theta,phi), phi);

subst( [theta=%pi/4, phi =0], Z1);
subst( [theta=%pi/4, phi =0], Z2);
```

▷ Click here to RUN the code.

MAXIMA output:

```
[%i2] Z1 : diff( R(theta,phi), theta);
[%o2]                [ - cos(phi) sin(theta), - sin(phi) sin(theta), cos(theta)]
[%i3] Z2 : diff( R(theta,phi), phi);
[%o3]                [ - sin(phi) cos(theta), cos(phi) cos(theta), 0]
[%i4] subst( [theta=%pi/4, phi =0], Z1);
[%o4]                [ - 1/sqrt(2), 0, 1/sqrt(2)]
[%i5] subst( [theta=%pi/4, phi =0], Z2);
[%o5]                [ 0, 1/sqrt(2), 0]
```

Remark. A tangent vector is a so-called *contravariant vector* and should conventionally better written in column form, e.g.

$$\mathbf{Z_1} = (-\cos(\phi)\sin(\theta), -\sin(\phi)\sin(\theta), \cos(\theta))^t = \begin{bmatrix} -\cos(\phi)\sin(\theta) \\ -\sin(\phi)\sin(\theta) \\ \cos(\theta) \end{bmatrix}$$

Admittedly, we've been a bit lax in this regard up until now.

3 Metric Tensor

The so-called *metric tensor*[6] g (or simply metric) plays an crucial role in Differential Geometry and General Relativity. It allows defining and calculate distances and angles on surfaces ('manifolds') in space, just as the inner product do it on a Euclidean space $\mathbb{R}^n$.

3.1 Definition: metric tensor

Let $R : U \to \mathbb{R}^n$ be the position vector aka a 'coordinate system' of a manifold M and differentiable on the open subset $U \subset \mathbb{R}^n$ with coordinates $(Z1, Z2, ..) \in U$.

a. Define the tangent vectors $\mathbf{Z}_i$ by differentiation of R w.r.t. each of the coordinates

$$\mathbf{Z}_i := \frac{\partial R}{\partial Z_i} \quad \overset{e.g.}{=} \overset{\text{Maxima}}{\text{diff}} \ (R(x,y),x) \tag{3.1}$$

These tangent vectors $(\mathbf{Z_1}, .., \mathbf{Z_n})$[7] are also called the *covariant basis* of the n-dimensional tangent space $T_P M$ with inner product $\bullet$.

b. Then, by definition, the (covariant) *metric tensor* $g = (g_{ij})$ consists of the pairwise dot products of the covariant basis vectors, i.e. for $i, j = 1, .., n$

$$g_{ij} := \mathbf{Z}_i \bullet \mathbf{Z}_j \tag{3.2}$$

The metric tensor is also called GRAM's matrix, therefore the notions g and g_{ij}.

c. The corresponding *contravariant metric tensor* $ug := g^{-1}$ is defined by $g^{ij} := (g^{-1})_{ij}$.

Remark.
∘ The metric tensor is heavily used to do so-called *index juggling*, cf. [7, p.88], i.e. doing the tensor operations of *raising* or *lowering* an index. In this context is is essential to keep track of the position of an index, i.e. if the index is *superscripted* ('upper index') or *subscripted* ('lower index') in a tensor expression. For this purpose, the terms *covariant* ('lower') resp. *contravariant* ('upper') index are introduced to distinguish between them and to use the EINSTEIN summation convention correctly.
∘ The following lexicon helps to memorize these conventions.

LEXICON	*Math*	MAXIMA `itensor` package
covariant tensor g of type (2,0):	g_{ij}	`g([i,j],[])`
contravariant tensor g of type (0,2):	g^{ij}	`g([],[i,j])`
mixed-variant tensor T of type (1,1):	T_i^j	`T([i],[j])`

∘ The components of a metric tensor g in a coordinate basis is a symmetric matrix (tensor), whose entries transform 'covariantly' under changes of the coordinate system. Thus a metric tensor is a 'covariant' symmetric tensor, cf.[47].

[6] aka *first fundamental form* or *fundamental tensor*
[7] We follow the notations in [7].

3.2 Examples

3.2.1 The Metric g in Cartesian coordinates in $\mathbb{R}^2$

The most elementary example is that of the two-dimensional Euclidean geometry, the Euclidean metric tensor g w.r.t. the canonical basis $E = (E_1, E_2) = (\begin{bmatrix}1\\0\end{bmatrix}, \begin{bmatrix}0\\1\end{bmatrix})$ with *cartesian coordinates*. Calculate g.

Solution: $\mathbf{1}^{st}$ mathematically.
We have

$$g_{11} = E_1 \bullet E_1 = 1,\ g_{12} = E_1 \bullet E_2 = 0,\ g_{21} = E_2 \bullet E_1 = 0,\ g_{22} = E_2 \bullet E_2 = 1$$

so we get the GRAM matrix alias the *metric tensor* g

$$g = (g_{ij}) = \begin{bmatrix}1 & 0\\ 0 & 1\end{bmatrix}$$

and their inverse matrix

$$g^{-1} = (g^{ij}) = \begin{bmatrix}1 & 0\\ 0 & 1\end{bmatrix} =: ug$$

$\mathbf{2}^{nd}$ use plain MAXIMA to look behind the curtain of `ctensor` used in 3^{rd}.

```
R(x,y) := [x,y];     /* identity mapping i.e. canonical coord.sys. */
Z1 : diff(R(x,y), x);
Z2 : diff(R(x,y), y);
Z  : matrix(Z1,Z2);

g11 : Z1.Z1;
g12 : Z1.Z2;
g21 : Z2.Z1;
g22 : Z2.Z2;

g  : matrix([g11,g12],[g21,g22]);
ug : invert(g);
```

▷ Click here to RUN the code.

$\mathbf{3}^{rd}$ use package `ctensor` and the predefined coordinate system `cartesian2d`.

```
/* basis in Euclidean vector space R^2 */
load(ctensor);                    /* 1 */
cframe_flag : true;              /* 2 */
ct_coordsys(cartesian2d);       /* 3 */
cmetric();                     /* 4 */
lg;                           /* 5 */
ug : invert(lg);             /* 6 */
lg.ug;                      /* 7 */
fr;  /* the contravariant Cartesian coordinate components */
```

Comment. In line 1: we load the tensor package `ctensor.mac`[8]. to do component tensor calculations. 2: computes the frame matrix 'fr' $= g$ and its inverse frame metric 'ufg'$= ug$. 3: sets up the predefined '*cartesian2d*' coordinate system, the dim=2 and the 4: metric tensor. 5: shows the lower metric tensor `lg`:$= (g_{ij}) = g$. 6: inverts the metric g to get the inverse, the upper metric tensor `ug`:$= (g^{ij}) = g^{-1}$. In 7: we verify, that the matrix product of g and her inverse $ug = g^{-1}$ gives the identity matrix E.

▷ Click here to RUN the code. YAMWI output:

```
[%i5] /* 4 */
      lg;
[%o5] matrix([1,0],[0,1])
[%i6] /* 5 */
      ug : invert(lg);
[%o6] matrix([1,0],[0,1])
[%i7] /* 6 */
      lg.ug;
[%o7] matrix([1,0],[0,1])
[%i8] /* 7 */
      fr;
[%o8] matrix([1,0],[0,1])
```

$\mathbf{4}^{th}$ use package `ctensor` and a user-defined coordinate system via `ct_coordsys`.

```
load("ctensor");
cframe_flag:true;
ct_coordsys([x,y, [x,y]]);  /* analog to R(x,y) := [x,y] in 1st */
ct_coords;
cmetric();
lg;
ug : invert(lg);
```

▷ Click here to RUN the code. YAMWI output:

```
[%i3] ct_coordsys([x,y, [x,y]]);
[%o3]                         done
[%i4] ct_coords;
[%o4]                        [x, y]
[%i5] cmetric();
[%o5]                        false
[%i6] lg;
[%o6]                   matrix([1,0],[0,1])
[%i7] ug : invert(lg);
[%o7]                   matrix([1,0],[0,1])
```

[8]These tensor packages are enhanced, streamlined and supported by Viktor TOTH, which is great work.

3.2.2 Affine coordinates in $\mathbb{R}^2$

From GRINFELD [7, p.64]. Suppose we have affine coordinates with covariant basis $Z_1 = \begin{bmatrix}2\\0\end{bmatrix}$ and $Z_2 = (\cos(\frac{\pi}{3}), \sin(\frac{\pi}{3}))^t = \begin{bmatrix}\frac{1}{2}\\ \frac{\sqrt{3}}{2}\end{bmatrix}$
Calculate the metric tensor and ist inverse.

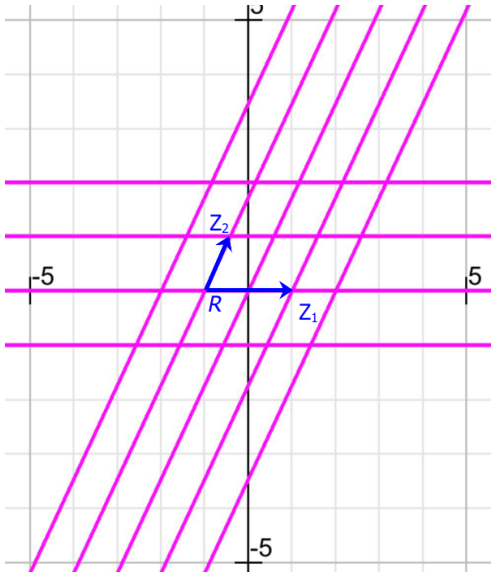


Figure 8: $R(x=-1, y=0)$: position vector in ambient $\mathbb{R}^2$.
Z_1: 1st covariant basis vector at R.
Z_2: 2nd covariant basis vector at R.
///..: the coordinate grid corresponding to basis Z.

Solution.

1st mathematically using Calculus.
We have

$$g_{11} = Z_1 \bullet Z_1 = |Z_1|^2 = 4,\ g_{12} = Z_1 \bullet Z_2 = 1,\ g_{12} = Z_1 \bullet Z_2 = 1,\ g_{12} = Z_1 \bullet Z_2 = 1$$

so we get the GRAM matrix g

$$g = (g_{ij}) = \begin{bmatrix}4 & 1\\ 1 & 1\end{bmatrix}$$

and their inverse matrix

$$ug := g^{-1} = (g^{ij}) = \begin{bmatrix}\frac{1}{3} & \frac{-1}{3}\\ \frac{-1}{3} & \frac{4}{3}\end{bmatrix}$$

BTW, hence, in anticipation of §4.1, we get the so-called *contravariant* basis vectors $\mathbf{Z}^1$, $\mathbf{Z}^2$ via

$$\begin{aligned} Z^1 &:= \frac{1}{3}Z_1 - \frac{1}{3}Z_2 = \begin{bmatrix} \frac{1}{2} \\ \frac{-1}{2\sqrt{3}} \end{bmatrix} \\ Z^2 &:= -\frac{1}{3}Z_1 + \frac{4}{3}Z_2 = \begin{bmatrix} 0 \\ \frac{2}{\sqrt{3}} \end{bmatrix} \end{aligned}$$

$\mathbf{2}^{nd}$ use MAXIMA.

```
/*-- we introduce affine coordinates in R^2 by */
Z1 : [2,0];
Z2 : [cos(%pi/3),sin(%pi/3)];

/*-- the covariant basis in affine coordinates is */
Z : matrix(Z1,Z2);

/*-- the GRAM matrix aka the metric tensor */
g : zeromatrix(2,2);
dim : 2;
for i thru dim do  for j thru dim do  g[i,j] : Z[i].Z[j];
g;

/*-- Calculate uZij, i.e. the inverse of Zij */
ug : invert(g);

/*-- the contravariant basis in affine coordinates is */
uZ1 : 1/3*Z1 -1/3*Z2;    /* ug[1] . transpose(Z1); */
uZ2 : -1/3*Z1 +4/3*Z2;

/* check orthogonality: */
Z1 . uZ1;  /* .. */
```

▷ Click here to RUN the code. YAMWI output:

```
[%i8]  /* Calculate uZij, i.e. the inverse of Zij */
       uZij : invert(Zij);
```

$$\begin{pmatrix} \frac{1}{3} & -\frac{1}{3} \\ -\frac{1}{3} & \frac{4}{3} \end{pmatrix}$$

```
[%i9]  /* the contrvariant basis in affine coordinates is */
       uZ1 : 1/3*Z1 -1/3*Z2;
```

$$\left[\frac{1}{2}, -\frac{1}{2\sqrt{3}}\right]$$

```
[%i10] /* uZij[1] . transpose(Z1); */
       uZ2 : -1/3*Z1 +4/3*Z2;
```

$$\left[0, \frac{2}{\sqrt{3}}\right]$$

```
[%i11] /* check: */
       Z1 . uZ1;
```

$$1$$

3.2.3 The metric tensor g for the polar coordinate system

The metric tensor g for the polar coordinate system in $\mathbb{R}^2$ is calculated with MAXIMA as follows:

```
/*-- we introduce polar coordinates in R^2 by */
define( R(r,theta) , [r*cos(theta),r*sin(theta)]);

/*-- the covariant basis in polar coordinates */
Z1 : diff( R(r,theta), r);
Z2 : diff( R(r,theta), theta);
Z  : matrix(Z1, Z2);

/*-- the covariant metric tensor */
U : [r,theta];
gij(i,j) := diff(R(r,theta), U[i]) . diff(R(r,theta), U[j])$      /* 1 */
g : matrix( [gij(1,1), gij(1,2)], [gij(2,1), gij(2,2)] );
g : trigsimp(g);

/*-- Calculate ug, i.e. the inverse of g */
/*   aka the contravariant metric Tensor     */

ug : invert(g);
```

Comment. In line 1: we use the coordinates $U = (r, \theta)$ to express the scalar products of the basis vectors. By definition of U, the numbers $U[i]$ resp. $U[j]$ are simply r resp. θ.

▷ Click here to RUN the code.
YAMWI output:

```
[%i5]  Z    : matrix(Z1, Z2);
[%o5]  ( cos(θ)     sin(θ)    )
       ( -r sin(θ)  r cos(θ)  )
[%i6]  /* covariant Metrik Tensor */
       U : [r,theta];
[%o6]  [r, θ]
[%i7]  gij(i,j) := diff(R(r,theta), U[i]) . diff(R(r,theta), U[j])$
[%i8]  g : matrix( [gij(1,1),gij(1,2)], [gij(2,1),gij(2,2)] );
[%o8]  ( sin^2(θ) + cos^2(θ)        0          )
       (          0          r^2 sin^2(θ) + r^2 cos^2(θ) )
[%i9]  g : trigsimp(g);
[%o9]  ( 1  0   )
       ( 0  r^2 )
[%i10]
[%o10] ( 1  0   )
       ( 0  1/r^2 )
```

3.2.4 The metric tensor g for the cylindrical coordinate system

The metric tensor in the cylinder coordinate systems in $\mathbb{R}^3$ can be calculated as follows, cf. [7, p.66, p.179]. Align a background Cartesian grid (x, y, z) with the cylindrical coordinates (r, θ, z). The position vector R, pointing to a position on the cylinder, is then given by

$$R(r, \theta, z) = (r\cos(\theta), r\sin(\theta), z)$$

The covariant basis vectors $\mathbf{Z}_i$ are the tangential vectors to the three coordinate lines, i.e. the covariant basis is obtained by partial differentiation of R with respect to r, θ and z:

$$\mathbf{Z}_r = \frac{\partial R}{\partial r} \qquad \mathbf{Z}_\theta = \frac{\partial R}{\partial \theta} \qquad \mathbf{Z}_z = \frac{\partial R}{\partial z}$$

Remember: at each point $R(r, \theta, z)$ we get *another* 'new' basis $(\mathbf{Z}_r, \mathbf{Z}_\theta, \mathbf{Z}_z)$!

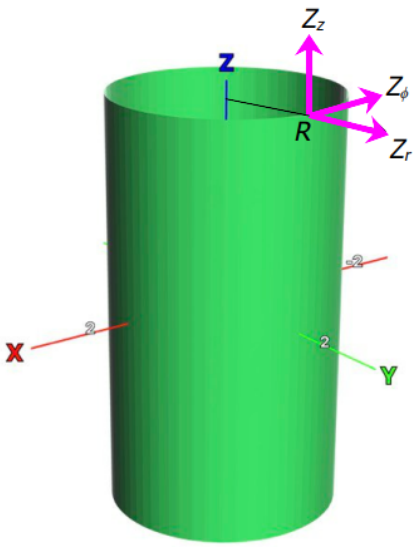


Figure 9: $R(r = 1, \theta = \pi/2, z = 2)$: position vector in $\mathbb{R}^3$.
Z_r: 1st covariant basis vector at R.
Z_θ: 2nd covariant basis vector at R.
Z_z: 3rd covariant basis vector at R.

Task: calculate the metric tensor g and its inverse g^{-1} for the cylindrical coordinate system.

We skip the mathematical calculation using Calculus (left to the reader) and do the Mathematics directly in MAXIMA.

1st We calculate the metric tensor using plain MAXIMA.

```
R(r, theta, z) := [r*cos(theta), r*sin(theta), z];
U : [r,theta,z];  /* the cylindrical coordinates */

g(i,j) := diff(R(r,theta,z), U[i]) . diff(R(r,theta,z), U[j])$   /* 1 */
g : genmatrix (lambda ([i, j], g(i,j)), 3, 3);                  /* 2 */

g  : trigsimp(g) ;
ug : invert(g);
```

Comment. 1: determines the partial differentiations of R with respect to r, θ and z and the corresponding scalar products. 2: generates the metric g represented as 3-by-3-matrix using the term $(i,j) \mapsto g(i,j)$ to calculate the entry (i,j) of g.

▷ Click here to RUN the code.
YAMWI output:

```
[%i4] g : genmatrix (lambda ([i, j], g(i,j)), 3, 3);
```

$$[\%o4]\quad \begin{pmatrix} \sin^2\theta+\cos^2\theta & 0 & 0 \\ 0 & r^2\sin^2\theta+r^2\cos^2\theta & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

```
[%i5] g : trigsimp(g) ;
```

$$[\%o5]\quad \begin{pmatrix} 1 & 0 & 0 \\ 0 & r^2 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

```
[%i6]
```

$$[\%o6]\quad \begin{pmatrix} 1 & 0 & 0 \\ 0 & \frac{1}{r^2} & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

2nd We calculate the metric tensor using `ctensor`.

```
load(ctensor);
cframe_flag : true;
ct_coordsys(polarcylindrical);      /* 1 */
cmetric();
lg : trigsimp(lg);                  /* 2 */
ug : trigsimp(invert(lg));          /* 3 */
fr : trigsimp(fr);                  /* 4 */

ldisplay(dim);                  /* 5 */
ldisplay(ct_coords);
ldisplay(lfg);
ldisplay(fri);
```

Remark. 1: sets up the *(polar)cylindrical* coordinate system. 2: gives the 'lower' (e.g. `g([i,j][])`$\equiv g_{ij}$) noted metric `lg` $= g$, which is inverted and then simplified with trigonometric formulas. 4: the 'fr' identifier of `ctensor` represents the *contravariant* Cartesian

coordinate components of a set of basis vectors, aligned with cylindrical coordinates in the Euclidean plane. In 5: we list some other objects of `ctensor`, e.g. the dimension of the space, the coordinates names and the inverted frame basis.

▷ Click here to RUN the code.

YAMWI output:

```
[%i5] lg : trigsimp(lg);
```

$$[\%o5]\quad \begin{pmatrix} 1 & 0 & 0 \\ 0 & r^2 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

```
[%i6]
```

$$[\%o6]\quad \begin{pmatrix} 1 & 0 & 0 \\ 0 & \frac{1}{r^2} & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

```
[%i7] /* 3 */
      fr : trigsimp(fr);
```

$$[\%o7]\quad \begin{pmatrix} \cos\theta & -\frac{\sin\theta}{r} & 0 \\ \sin\theta & \frac{\cos\theta}{r} & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

Exercise 7. Use the following *user-defined* coordinate system, which explicit sets up the cylindrical coordinates and repeat the calculation from above:

```
load("ctensor");
cframe_flag : true;
ct_coordsys( [r*cos(phi), r*sin(phi), z,  [r,phi,z]]);
```

▷ Click here to START your solution.

Compare with the code in 1^{st}, in particular with the definition of R and U.

4 Contravariant Basis

With the help of the tangent vectors $\mathbf{Z}_i$ and the covariant metric tensor $g = (g_{ij})$ we define the so-called *contravariant* basis $\mathbf{Z}^i$. These objects also play a major role in Tensor Analysis and Differential Geometry.

4.1 Definition: the contravariant basis

Let $R : U \to \mathbb{R}^n$ be the position vector of a manifold M and differentiable on the open subset $U \subset \mathbb{R}^n$ with coordinates $(Z1, Z2, ..) \in U$.
Then we defined in §3.1

- the tangent vectors $\mathbf{Z}_i := \frac{\partial R}{\partial Z_i}$ by differentiation of R w.r.t. each of the coordinates, also called the *covariant basis* vectors of the tangent space $T_P M$.
- the covariant *metric tensor* g by $g_{ij} := \mathbf{Z}_i \bullet \mathbf{Z}_j$
- the *contravariant metric tensor* $ug := g^{-1}$ by $g^{ij} := (g^{-1})_{ij}$.

Now we define the ith *contravariant basis* vector $\mathbf{Z}^i$ by the linear combination[9]

$$\mathbf{Z}^i := g^{ij} \cdot \mathbf{Z}_i \overset{Einstein}{=} \sum_{i=1}^{n} g^{ij} * \mathbf{Z}_i \tag{4.1}$$

Remark. Remember the EINSTEIN summation convention: The **co**variant tangent vectors $\mathbf{Z}_i$ are written with one index as **sub**script, the corresponding **contra**variant basis vector $\mathbf{Z}^i$ with a **super**script index. Then a summation in such a tensor expression is implicit done, when an index appears twice, once as a subscript ('lower index') and once as a superscript ('upper index').

4.2 Examples

4.2.1 The contravariant basis for polar coordinate system

We introduce polar coordinates in $\mathbb{R}^2$ by the position vector

$$R(r, \theta) := (r\cos(\theta), r\sin(\theta))$$

Then the covariant basis $\mathbf{Z}$ in polar coordinates is calculated by[10]

$$\begin{aligned} \mathbf{Z}_1 &:= \frac{\partial R}{\partial r} = (\cos(\theta), \sin(\theta)) \\ \mathbf{Z}_2 &:= \frac{\partial R}{\partial \theta} = (-r\sin(\theta), r\cos(\theta)) \\ \mathbf{Z} &:= (\mathbf{Z_1}, \mathbf{Z_2}) \end{aligned}$$

[9] cf. [7, p.58]
[10] sometimes we write $\mathbf{Z}_r$ or E_r instead of $\mathbf{Z}_1$ and $\mathbf{Z}_\theta$ or E_θ for $\mathbf{Z}_2$, see fig. 10.

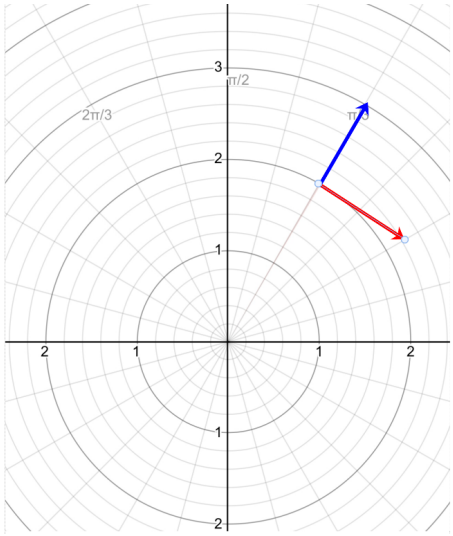


Figure 10: Z_r: 1st covariant basis vector at $R(r = 2, \theta = \pi/3)$.
Z_θ: 2nd covariant basis vector at R.

We determine the *contravariant* basis in three ways.

1st we calculate the contravariant basis **uZ** using Calculus.

First we calculate the metric tensor g:

$$\begin{aligned}
g_{11} := \mathbf{Z_1} \bullet \mathbf{Z_1} &= (\cos(\theta), \sin(\theta)) \bullet (\cos(\theta), \sin(\theta)) = \cos(\theta)^2 + \sin(\theta)^2 = 1 \\
g_{12} := \mathbf{Z_1} \bullet \mathbf{Z_2} &= (\cos(\theta), \sin(\theta)) \bullet (-r\sin(\theta), r\cos(\theta)) = 0 \\
g_{21} := \mathbf{Z_2} \bullet \mathbf{Z_1} &= (-r\sin(\theta), r\cos(\theta)) \bullet (\cos(\theta), \sin(\theta)) = 0 \\
g_{22} := \mathbf{Z_2} \bullet \mathbf{Z_2} &= (-r\sin(\theta), r\cos(\theta)) \bullet (-r\sin(\theta), r\cos(\theta)) = r^2 \\
g &= \begin{bmatrix} 1 & 0 \\ 0 & r^2 \end{bmatrix}
\end{aligned}$$

Next we calculate the inverse metric tensor

$$g^{-1} = \begin{bmatrix} 1 & 0 \\ 0 & \frac{1}{r^2} \end{bmatrix} = \begin{bmatrix} g^{11} & g^{12} \\ g^{21} & g^{22} \end{bmatrix}$$

and get the contravariant basis p.def.

$$\begin{aligned}
\mathbf{Z^1} &= g^{11}\mathbf{Z_1} + g^{12}\mathbf{Z_2} \overset{Einstein}{\equiv} g^{1i}\mathbf{Z_i} = 1\cdot(\cos(\theta),\sin(\theta)) + 0\cdot(-r\sin(\theta), r\cos(\theta)) \\
&= (\cos(\theta),\sin(\theta)) \\
\mathbf{Z^2} &= g^{21}\mathbf{Z_1} + g^{22}\mathbf{Z_2} \overset{Einstein}{\equiv} g^{2i}\mathbf{Z_i} = 0\cdot(\cos(\theta),\sin(\theta)) + \frac{1}{r^2}\cdot(-r\sin(\theta), r\cos(\theta)) \\
&= (-\frac{\sin(\theta)}{r}), \frac{\cos(\theta)}{r}) \\
\mathbf{uZ} &:= (\mathbf{Z^1}, \mathbf{Z^2})
\end{aligned}$$

This calculation is a little expensive, so we
$\mathbf{2}^{nd}$ calculate the contravariant basis also with plain MAXIMA.

```
R(r, theta) := [r*cos(theta),r*sin(theta)];
Z1 : diff(R(r, theta), r);
Z2 : diff(R(r, theta), theta);
Z : matrix(Z1,Z2);

g11 : trigsimp(Z1.Z1);
g12 : Z1.Z2;
g21 : Z2.Z1;
g22 : trigsimp(Z2.Z2);

g : matrix([g11,g12],[g21,g22]);
ug : invert(g);

uZ1 : ug[1,1]*Z1 + ug[1,2]*Z2;
uZ2 : ug[2,1]*Z1 + ug[2,2]*Z2;
/* or do */
uZ1 : row(ug, 1) . Z;
uZ2 : row(ug, 2) . Z;
```

▷ Click here to RUN the code.

YAMWI output:

```
[%i10] ug : invert(g);
```

$$\begin{pmatrix} 1 & 0 \\ 0 & \frac{1}{r^2} \end{pmatrix}$$

```
[%i11] uZ1 : ug[1,1]*Z1 + ug[1,2]*Z2;
```

$$[\cos\theta, \sin\theta]$$

```
[%i12] uZ2 : ug[2,1]*Z1 + ug[2,2]*Z2;
```

$$\left[-\frac{\sin\theta}{r}, \frac{\cos\theta}{r}\right]$$

```
[%i13] uZ1 : row(ug, 1) . Z;
```

$$\begin{pmatrix} \cos\theta & \sin\theta \end{pmatrix}$$

```
[%i14] uZ2 : row(ug, 2) . Z;
```

$$\begin{pmatrix} -\frac{\sin\theta}{r} & \frac{\cos\theta}{r} \end{pmatrix}$$

3rd We now calculate the contravariant basis alternatively using `ctensor` package.

- **Remark.** To discriminate between the covariant resp. contravariant basis we used the words 'co..' resp. 'contra..' and subscript resp. superscript indices in the mathematical notation. Using plain MAXIMA we used the attributes 'l..' resp. 'u..' for lower resp. upper position of the indices. But this is cumbersome. Therefore one use special packages like `ctensor` or `itensor` to reflect that logic by means of index notation in brackets: `[`$_{covariant}$`][`^{contravariant}`]`, i.e. in the first `[..]` we collect all covariant indices and in the second `[..]` we collect all contravariant inidces.

- To sum up using the metric tensor as example:

LEXICON	*Math*	MAXIMA i/c/tensor syntax
		covariant
covariant metric	g_{ij}	`g([i,j],[])`
contravariant metric	g^{ij}	`g([],[i,j])`
		contravariant

Therefore, using the indicial tensor package `itensor` we have

```
load(itensor);
("tensors are _i_ndexed objects. To show a tensor use ishow()")$
ishow(g([a,b],[]))$
ishow(g([],[i,j]))$
```

YAMWI output:

```
[%i3] ishow(g([a,b],[]))$
[%o3]                                  g
                                        a b
[%i4] ishow(g([],[i,j]))$
[%o4]                                  ij
                                       g
```

▷ Click here to RUN the code.

◦ We now calculate the *contravariant basis* using the component tensor package `ctensor`.

```
load(ctensor);                          /* 1 */
cframe_flag : true;                    /* 2 */
ct_coordsys(polar);                   /* 3 */
cmetric();                           /* 4 */
lg;                                 /* 5 */
trigsimp(lg);                      /* 6 */
ug : trigsimp(invert(lg));        /* 7 */
fr;                              /* 8 */
trigsimp(fr);                   /* 9 */
```

Comment. To get used to the tensor package `ctensor`, we'll comment once again. In line 1: we load the tensor package `ctensor.mac` to do component tensor calculations. 2: computes the frame matrix 'fr' $= g$ and its inverse frame metric 'ufg'$= ug$. 3: sets up the predefined '*polar*' coordinate system and its 4: metric tensor. 5: gives the lower metric tensor $g = (g_{ij})$, which is simplified in 6: using trigonometric formulas. In 7: we invert the covariant metric $g = (g_{ij})$ and get her inverse $ug = (g^{ij})$. In 8: we calculate the contravariant basis vectors, which we pick as columns of this matrix representation.

▷ Click here to RUN the code.

YAMWI output:

```
[%i5] /*lg;*/
      trigsimp(lg);
```

$$[\%o5]\quad \begin{pmatrix} 1 & 0 \\ 0 & r^2 \end{pmatrix}$$

```
[%i6]
```

$$[\%o6]\quad \begin{pmatrix} 1 & 0 \\ 0 & \frac{1}{r^2} \end{pmatrix}$$

```
[%i7] fr;
```

$$[\%o7]\quad \begin{pmatrix} \frac{\cos\varphi\, r}{\sin^2\varphi\, r+\cos^2\varphi\, r} & -\frac{\sin\varphi}{\sin^2\varphi\, r+\cos^2\varphi\, r} \\ \frac{\sin\varphi\, r}{\sin^2\varphi\, r+\cos^2\varphi\, r} & \frac{\cos\varphi}{\sin^2\varphi\, r+\cos^2\varphi\, r} \end{pmatrix}$$

```
[%i8] /* represents the contravariant Cartesian coordinate components */
      trigsimp(fr);
```

$$[\%o8]\quad \begin{pmatrix} \cos\varphi & -\frac{\sin\varphi}{r} \\ \sin\varphi & \frac{\cos\varphi}{r} \end{pmatrix}$$

Remark. The 1st contravariant basis vector $\mathbf{Z}^1$ in polar coordinates equals the first covariant basis vector $\mathbf{Z}_1$, whereas the 2nd contravariant basis vector $\mathbf{Z}^2$ is collinear with the second covariant basis vector and has length $|\mathbf{Z}^2| = 1/r$, see fig.10.

4.2.2 The contravariant basis on a sphere

The sphere S_r^2 with radius r is described by means of a parametrization using the position vector $R : U \to \mathbb{R}^3$ defined by

$$(r, \theta, \phi) \mapsto (r\sin(\theta)\cos(\phi), r\sin(\theta)\sin(\phi), r\cos(\theta))$$

which is differentiable on an open subset $U \subset \mathbb{R}^2$ with $(r, \theta, \phi) \in U$.

Task:

∘ Use the tensor package `ctensor` to calculate the covariant basis $(\mathbf{Z}_1, \mathbf{Z}_2, \mathbf{Z}_3)$ and the contravariant basis $(\mathbf{Z}^1, \mathbf{Z}^2, \mathbf{Z}^3)$[11] :
a. at a general position $P = R(r, \theta, \phi)$.
b. at the special position $P = R(2, 45°, 0)$.

Solution: Because the calculation is very laborious, we use right away the tensor package `ctensor`.

a. We formulate the position vector R as transformation rule using function `ct_coordsys()`.

```
load("ctensor");
cframe_flag : true;
ct_coordsys([r*sin(theta)*cos(phi),
             r*sin(theta)*sin(phi),
             r*cos(theta),          [r,theta,phi]]);
ct_coords;
fri;
Z1 : col(fri, 1);
Z2 : col(fri, 2);
Z3 : col(fri, 3);

cmetric();
lg : trigsimp(lg);       /* _l_ower g = g_{ij} = g([i,j],[])  */
ug : invert(lg);         /* _u_pper g = g^{ij} = g([],[i,j])  */
uZ1 : ug[1,1]*col(fri,1) + ug[1,2]*col(fri,2) + ug[1,3]*col(fri,3);
uZ2 : ug[2,1]*col(fri,1) + ug[2,2]*col(fri,2) + ug[2,3]*col(fri,3);
uZ3 : ug[3,1]*col(fri,1) + ug[3,2]*col(fri,2) + ug[3,3]*col(fri,3);
```

▷ Click here to RUN the code.

[11] the latter is denoted `(uZ1,uZ2,uZ3)` in MAXIMA script below, 'u' means 'upper' index.

YAMWI output:

```
[%i8]  /* Z : matrix(Z1,Z2); = lg*/
       lg:trigsimp(lg);
```

$$[\%o8]\quad \begin{pmatrix} 1 & 0 & 0 \\ 0 & r^2 & 0 \\ 0 & 0 & r^2\sin^2\theta \end{pmatrix}$$

```
[%i9]
```

$$[\%o9]\quad \begin{pmatrix} 1 & 0 & 0 \\ 0 & \frac{1}{r^2} & 0 \\ 0 & 0 & \frac{1}{r^2\sin^2\theta} \end{pmatrix}$$

```
[%i10] uZ1 : ug[1,1]*col(fri, 1) + ug[1,2]*col(fri, 2)+ug[1,3]*col(fri, 3);
```

$$[\%o10]\quad \begin{pmatrix} \cos\varphi\sin\theta \\ \sin\varphi\sin\theta \\ \cos\theta \end{pmatrix}$$

```
[%i11] uZ2 : ug[2,1]*col(fri, 1) + ug[2,2]*col(fri, 2)+ug[2,3]*col(fri, 3);
```

$$[\%o11]\quad \begin{pmatrix} \frac{\cos\varphi\cos\theta}{r} \\ \frac{\sin\varphi\cos\theta}{r} \\ -\frac{\sin\theta}{r} \end{pmatrix}$$

```
[%i12] uZ3 : ug[3,1]*col(fri, 1) + ug[3,2]*col(fri, 2)+ug[3,3]*col(fri, 3);
```

$$[\%o12]\quad \begin{pmatrix} -\frac{\sin\varphi}{r\sin\theta} \\ \frac{\cos\varphi}{r\sin\theta} \\ 0 \end{pmatrix}$$

b. Substituting the concrete values of $R = (2, 45°, 0) = (2, \frac{\pi}{4}, 0)$ into Z_1, Z_2, Z_3 we get the following figure:

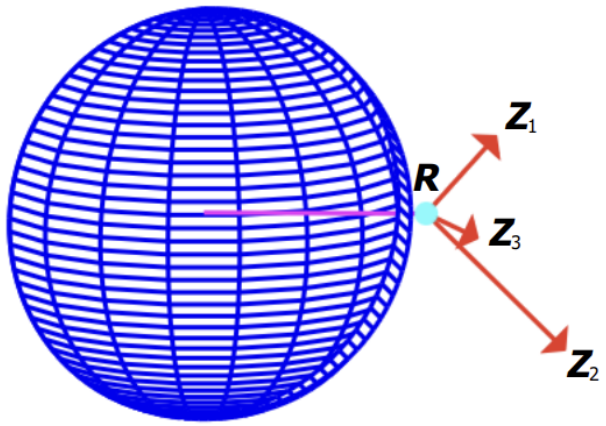


Figure 11: R: the position vector at $R(r = 2, \theta = \pi/4, \phi = 0)$.
Z_1, Z_2, Z_3: the covariant basis vectors at R.

◦ If you want to change the viewpoint or make other changes to the sketch, you find the source code here: ▷ Click here to RUN the code.[12]

[12] I am very grateful to Michael GOSSE for fixing my code.

5 What is a tensor?

From wikipedia: ▷ Tensor. 'Each type of tensor comes equipped with a transformation law that details how the components of the tensor respond to a change of basis. The components of a vector can respond in two distinct ways to a change of a basis, see covariance and contravariance of vectors.'

A very understandable decription is by V. Toth.

Here is the correct and precise mathematical Wikipedia definition of a tensor, in case you want to be shocked and will feel incomprehensible - we therefore cite this definition in tiny print, you may skip it:

5.1 Definition: Tensor

- (Wikipedia, ibid.) A *tensor of type* (p, q) is an assignment of a multidimensional array $T^{i_1\dots i_p}_{j_1\dots j_q}[\mathbf{f}]$ to each basis $\mathbf{f} = (\mathbf{e}_1, \dots, \mathbf{e}_n)$ of an n-dimensional vector space such that,

  if we apply the change of basis $\mathbf{f} \mapsto \mathbf{f} \cdot R = \left(\mathbf{e}_i R^i_1, \dots, \mathbf{e}_i R^i_n\right)$

  then the multidimensional array obeys the transformation law

$$T^{i'_1\dots i'_p}_{j'_1\dots j'_q}[\mathbf{f} \cdot R] = \left(R^{-1}\right)^{i'_1}_{i_1} \cdots \left(R^{-1}\right)^{i'_p}_{i_p} \quad T^{i_1,\dots,i_p}_{j_1,\dots,j_q}[\mathbf{f}] \quad R^{j_1}_{j'_1} \cdots R^{j_q}_{j'_q}.$$

Instead, we cite Peter Collier's description [5, p.140] to get a feeling what a tensor 'is':

- *a mathematical object* represented by a collection of components that transform a certain way.
- *a linear machine* that produces numbers when fed with covariant and contravariant vectors. Each lower index represent a slot that can be fed by a covariant vector, each upper index represents a slot that can be fed by a contravariant vector.
  When all the slots are filled, out pops a number.

Remark.

- V. Toth[13] gives the following overview for the representation of tensors:

Representation of a contravariant vector	$\begin{bmatrix}1\\2\end{bmatrix}$
Representation of a covariant vector	$[1 \quad 2]$
Representation of a (2,0) tensor	$[[1,2] \quad [3,4]]$
Representation of a (1,1) tensor	$\begin{bmatrix}1 & 2\\3 & 4\end{bmatrix}$
Representation of a (0,2) tensor	$\begin{bmatrix}\begin{bmatrix}1\\2\end{bmatrix}\\\begin{bmatrix}3\\4\end{bmatrix}\end{bmatrix}$
The Christoffel-symbol in 2D	$\Gamma^k_{ij} = \begin{bmatrix}[\Gamma^0_{00}\Gamma^0_{01}] & [\Gamma^0_{10}\Gamma^0_{11}]\\[\Gamma^1_{00}\Gamma^1_{01}] & [\Gamma^1_{10}\Gamma^1_{11}]\end{bmatrix}$

[13] in a private communication

◦ Toth distinguish three ways of looking at tensors: in terms of its representation (i.e. a matrix), in terms of the algebraic rules of matrix manipulation and third with focus on the purely algebraic properties of tensors, see ▷ Tensors.

◦ We often denote an Euclidean vector $v = (a_1, ..., a_n) \in \mathbb{R}^n$ simply as a list of entries, e.g. $v = (3, 2) \in \mathbb{R}^2$. In the language of tensors we should precise discriminate between a **co**variant vector $v = [3\ 2] = [v_1\ v_2]$ i.e. a tensor of type $(1, 0)$ with *one lower index* vs. a **contra**variant vector $v = \begin{bmatrix} v^1 \\ v^2 \end{bmatrix} = \begin{bmatrix} 3 \\ 2 \end{bmatrix}$ i.e. a tensor of type $(0, 1)$ with *one upper index* and reflect this distinction as far as possible in the description in the CAS.

◦ So we may say: a tensor with respect to a basis is represented by a multidimensional array. In a figurative rough sense, tensors 'are' 'multidimensional' matrices.

To make this complicated definition §5.1 more grounded, we will show some simple cases.

5.2 Example: This is _*not*_ a tensor!

We define and test an indexed object T, cf. the example by Grinfeld [7, p.94].

$$T_{ijk} := i^2jk$$

First we represent T with the CAS EigenMath, because the output is easy to interpret:

```
## EIGENMATH
i = quote(i)                    -- clear 'i', the imaginary unit ..

-- define and calculate elements of T as function values

Tijk(i,j,k) = i^2*j*k           -- would be -jk, if 'i' wasn't cleared
Tijk                            --(1)
Tijk(1,2,1)                     --(2) 1st matrix, 2nd row, 1st entry

--  T as multidimensional array

Tijk = zero(2, 2,2)             -- empty container for object T
for(i,1,2,                      -- three loops to fill container Tijk
  for(j,1,2,
    for(k,1,2,  Tijk[i,j,k] = i^2*j*k)))
Tijk                            -- (3)
Tijk[1,2,1]                     -- (4) 1st matrix, 2nd row, 1st entry
```

▷ Click here to Run the script in EigenMath.

EigenMath output:

$$T_{ijk} = i^2 j\, k$$

$$2$$

$$T_{ijk} = \begin{bmatrix} \begin{bmatrix} 1 & 2 \\ 2 & 4 \end{bmatrix} \\ \begin{bmatrix} 4 & 8 \\ 8 & 16 \end{bmatrix} \end{bmatrix}$$

Figure 12: 1: T defined as function $T_{ijk} := i^2 jk$.
2: value of T at index triple $(i, j, k) = (1, 2, 1)$ is 2.
3: T displayed as multidimensional array $[(::), (::)]$.

Comment. We define object T in two ways. First T is defined by computing each individual value by the term $i^2 jk$ in (1). Here you have to address the value via `Tijk (..)` for the function call. Alternatively T is defined in (2) as a multidimensional array whose values are read in with matrix-access brackets `Tijk[..]`. In this way the complete table of T-values is available at once.

Second we use the CAS Maxima^{online} to do the same job:[14]

```
/* MAXIMA */
/* define and calculate elements of T as function values */
Tijk(i,j,k) := i^2*j ;
Tijk(1,2,1) ;

/*  T as multidimensional 'matrix' -> array needed! */
dim : 2;
Tijk : make_array(any, dim, dim, dim)$                    /* (1) */
for i from 0 thru dim-1 do
    for j from 0 thru dim-1 do
        for k from 0 thru dim-1 do
              Tijk[i,j,k]: (i+1)^2 * (j+1) * (k+1);   /* (2) */
Tijk;                                                  /* (3) */
Tijk[0,1,0];                                          /* (4) */
```

▷ Click here to RUN the code in Maxima^{online}.

[14] I thank Michel Gosse for fixing an mistake.

YAMWI output:

```
[%i6] Tijk;
[%o6] {Lisp Array: #3A(((1 2) (2 4)) ((4 8) (8 16)))}
```

Comment. In (1) we create an Lisp array with type 'any'. There are indices, but the i'th index runs from 0 to $length(array) - 1$. Each individual T value is filled by the adapted term $(i+1)^2 * (j+1) * k$ with shifted values in (2). The output (%o6) displays the object T (array) line-oriented.

Third we try the `ctensor` package to do the same job:

```
load(ctensor);

for i thru dim do
      for j thru dim do
            for k thru dim do   T [i,j,k] : i^2*j*k;

T[1,2,1];
cdisplay(T);
```

▷ Click here to RUN the code in MAXIMA^{online}.

YAMWI output:

[%i3] `T[1,2,1];`

[%o3] 2

[%i4] `cdisplay(T);`

[%o4]
$$T_1 = \begin{pmatrix} 1 & 2 & 3 & 4 \\ 2 & 4 & 6 & 8 \\ 3 & 6 & 9 & 12 \\ 4 & 8 & 12 & 16 \end{pmatrix}$$

[%o4]
$$T_2 = \begin{pmatrix} 4 & 8 & 12 & 16 \\ 8 & 16 & 24 & 32 \\ 12 & 24 & 36 & 48 \\ 16 & 32 & 48 & 64 \end{pmatrix}$$

Comment. We see that `ctensor` interpret the term $T[i,j,k] : i^2 * j * k$ in another way, i.e. tensoriell. So the original object is _*not*_ a tensor. GRINFELD names such indexed objects *variant*s, because they does not follow the transformation rules. When the same rule is applied in different coordinate systems, the results of a variant differ, i.e. they are not the same. *Not a tensor, but only a variant*!

Exercise 8. (Variant T_{ijkl} with prescribed values.)

$$T_{ijkl} = \begin{bmatrix} \begin{bmatrix} 1 & 2 \\ 2 & 4 \end{bmatrix} & \begin{bmatrix} 2 & 4 \\ 4 & 8 \end{bmatrix} \\ \begin{bmatrix} 4 & 8 \\ 8 & 16 \end{bmatrix} & \begin{bmatrix} 8 & 16 \\ 16 & 32 \end{bmatrix} \end{bmatrix}$$

1. Determine the function and the table-oriented formula for the given variant T.
2. Do access for the value 16 in the 3th matrix inside variant T.
3. Represent T in MAXIMA^{online} and using `ctensor`. Think about the results.

Solution:

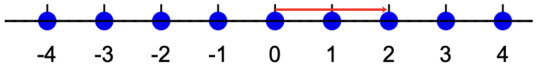
1. We give the defining relation as function value, ▷ Click here to see the solution.
2. We define T as a tensor 'table', ▷ Click here to look at the ssolution.
3. to be left as an exercise.

▷ Click here to do the exercise.
You should see the variant T as shown in the beginning of the exercise.

5.3 Examples: These _*are*_ tensors

5.3.1 1D scaling transformation

In a one dimensional space (imagine the number line), a scaling transformation is a linear operation that stretches or compresses all points relative to a fixed origin 0. The coordinate system is changed by e.g. doubling the length of the unit, i.e. changing the unit from 1-cm to 2-cm units.



1. *Visualization*: Imagine a number line labeled with markers at $..-3,-2,-1,0,1,2,3,..$ If you apply a scale factor of $s = 2$, the marker at 1 moves to 2, 2 moves to 4, -1 moves to -2. The space expands outward from zero.
   If you apply s=0.5, the space shrinks toward zero.

2. *Coordinate transformation*: Every vector (a number) x is mapped to a new number x' by a scale factor $s = 2$: $x' = s \cdot x$. The new coordinate x' is double of the old coordinate x.

3. *Jacobian*: $J_x^{x'} := \frac{\partial x'}{\partial x} = \frac{\partial}{\partial x}(2x) = 2 = s$.

4. *Vector transformation*: Let the one dimensional vector (number) be $v = 3$ units in the old system, then $v' = J_x^{x'} \cdot v = 2 \cdot 3 = 6$.
   In the new system the vector v has new component $v' = 6$.

We now use plain MAXIMA .

```
R(x) := [2*x];            /* position vector i.e. coordinate change */
Z : diff(R(x), x);       /* the Jacobian, i.e. the covariant basis */
g11 : Z . Z;
g   : matrix([g11]);
ug  : invert(g);
uZ  : ug*Z;          /* the contravariant basis */
```

▷ Click here to RUN the code in MAXIMA^{online}. YAMWI output:

```
[%i1] /* position vector i.e. coordinate change */
      R(x) := [2*x];
[%o1]                    R(x) := [2 x]
[%i2] /* the Jacobian, i.e. covariant basis */
      Z : diff(R(x), x);
[%o2]                          [2]
[%i3] g11 : Z . Z;
[%o3]                           4
[%i4] g : matrix([g11]);
[%o4]                          ( 4 )
[%i5] ug : invert(g);
[%o5]                          1/4
[%i6] /* the contravariant basis */
      uZ : ug*Z;
[%o6]                          [1/2]
```

Example 6. The vectors of the covariant basis $(\mathbf{Z}_1, \mathbf{Z}_2, \mathbf{Z}_3)$ and the contravariant basis $(\mathbf{Z}^1, \mathbf{Z}^2, \mathbf{Z}^3)$ are tensors. The proof uses the chain rule and is shown e.g. in [7, p.77]. The result is: if we relate $\mathbf{Z}_{i'}$ with its primed coordinates and $\mathbf{Z}_i$ with the unprimed, we get

$$\mathbf{Z}_{i'} = \mathbf{Z}_i \cdot J^i_{i'}$$

where $J^i_{i'}$ is the so-called *Jacobian* of the coordinate transformation defined by $J^i_{i'} := \frac{\partial Z^i(Z')}{\partial Z^{i'}}$. See V. Toth in ▷ On tensors and their matrix representations.

5.3.2 Example: the line element ds^2 transforms *contravariantly*

In the following example, we want to show that *a vector, viewed as a tensor*, is more than just an object with magnitude and direction. In particular, we show that the *line element* ds^2 transforms *contravariantly* from one coordinate system to another.

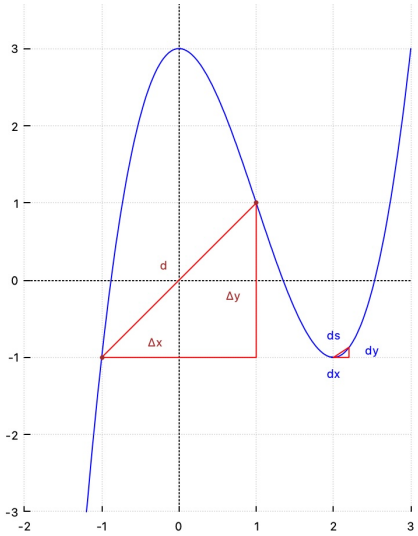


Figure 13: △: distance between two points via $d^2 = (\triangle x)^2 + (\triangle y)^2$
△: and two infinitesimal close points via $(ds)^2 = (dx)^2 + (dy)^2$

To calculate the squared distance d^2 between the two points $P(-1,-1) = (x^1, y^1)$ and $Q(1,1) = (x^2, y^2)$ we do $d^2 = (x^2 - x^1)^2 + (y^2 - y^1)^2 = (1+1)^2 + (1+1)^2 = 8$, ergo $d = 2\sqrt{2}$. This motivates the definition of the *line element* ds resp. its square ds^2 as the distance of two infinitesimal close points with tiny displacements dx, dy in a two dimensional space:

$$ds^2 = dx^2 + dy^2$$

This definition extends immediately to higher dimensions.
Looking at the multiplication table of the infinitesimal displacements

$$g = (g_{ij}) = \begin{array}{c|cc} - & dx & dy \\ dx & 1 & 0 \\ dy & 0 & 1 \end{array}$$

we may accept, that the coefficients of the metric tensor g_{ij} are often described using the line element ds^2.

Task: Find the line element and the metric of $\mathbb{R}^2$ in the polar coordinate system.

Solution:

1st We know that $x = r\cos(\theta) =: x(r,\theta)$ and $y = r\sin(\theta) =: y(r,\theta)$ depends explicit of r and θ . Therefore, on an infinitesimal small scale, the differentials dx resp. dy depend linearly on the differentials of r and θ.
With the abbreviation $ds^2 \equiv (ds)^2, dx^2 \equiv (dx)^2, dr^2 \equiv (dr)^2$ etc. we argue

$$\begin{aligned}
dx &= \cos(\theta)\cdot dr - r\cdot\sin(\theta)\cdot d\theta \\
dy &= \sin(\theta)\cdot dr + r\cdot\cos(\theta)\cdot d\theta \\
ds^2 &= dx^2 + dy^2 \\
&= (\cos(\theta)\cdot dr - r\cdot\sin(\theta)\cdot d\theta)^2 + (\sin(\theta)\cdot dr + r\cdot\cos(\theta)\cdot d\theta)^2 \\
&= ... \\
&= \cos(\theta)^2\cdot dr^2 + r^2\cdot\sin(\theta)^2\cdot d\theta^2 + \sin(\theta)^2\cdot dr^2 + r^2\cdot\cos(\theta)^2\cdot d\theta^2 \\
&= dr^2 + r^2\cdot d\theta^2
\end{aligned}$$

From the line element $ds^2 = 1\cdot dr^2 + r^2\cdot d\theta^2$ we may read off the metric tensor

$$g = (g_{ij}) = \begin{bmatrix} 1 & 0 \\ 0 & r^2 \end{bmatrix}$$

2nd We transform the line element ds^2 in $\mathbb{R}^2$ *from polar* (r,θ) *to Cartesian* (x,y) coordinates a little more abstract in order to show the transformation rule.
From 1^{st} we get

1. $dx = \frac{\partial x}{\partial r}dr + \frac{\partial x}{\partial \theta}d\theta$ because x depends on r and θ
   $dy = \frac{\partial y}{\partial r}dr + \frac{\partial y}{\partial \theta}d\theta$ also $y(r,\theta)$ is function of r and θ
2. $\begin{bmatrix} dx \\ dy \end{bmatrix} = \begin{bmatrix} \frac{\partial x}{\partial r} & \frac{\partial x}{\partial \theta} \\ \frac{\partial y}{\partial r} & \frac{\partial y}{\partial \theta} \end{bmatrix} \bullet \begin{bmatrix} dr \\ d\theta \end{bmatrix}$ write as matrix equation

   To get a tensorial notation of 1. resp. 2. we switch to index notation using the lexicon $\begin{smallmatrix} x \\ y \end{smallmatrix} \mid \begin{smallmatrix} \bar{x}^1 \\ \bar{x}^2 \end{smallmatrix}$ for primed (bar'ed) coordinates and unprimed (unbar'ed) coordinates $\begin{smallmatrix} r \\ \theta \end{smallmatrix} \mid \begin{smallmatrix} x^1 \\ x^2 \end{smallmatrix}$. Now 2. is written as

3. $\begin{bmatrix} d\bar{x}^1 \\ d\bar{x}^2 \end{bmatrix} = \begin{bmatrix} \frac{\partial \bar{x}^1}{\partial x^1} & \frac{\partial \bar{x}^1}{\partial x^2} \\ \frac{\partial \bar{x}^2}{\partial x^1} & \frac{\partial \bar{x}^2}{\partial x^2} \end{bmatrix} \bullet \begin{bmatrix} dx^1 \\ dx^2 \end{bmatrix}$ write with un/primed coordinates
4. $d\bar{x}^1 = \sum_{k=1}^{2} \frac{\partial \bar{x}^1}{\partial x^k}\cdot dx^k, \quad d\bar{x}^2 = \sum_{k=1}^{2} \frac{\partial \bar{x}^2}{\partial x^k}\cdot dx^k$ write down as 2 individual equations

5. $d\bar{x}^j = \sum_{k=1}^{2} \frac{\partial \bar{x}^j}{\partial x^k} \cdot dx^k$ write as *one* compact equation using free index $j = 1, 2$

6.
$$d\bar{x}^j = \frac{\partial \bar{x}^j}{\partial x^k} \cdot dx^k$$
even more compact using EINSTEIN summation convention

In summa: the tensor equation 6. is equivalent to the *both* equations 3. resp. 4.
If we rename $\frac{\partial \bar{x}^j}{\partial x^k} =: a_{jk}$ and $d\bar{x}^j =: \bar{T}^j$ resp. $dx^j =: T^j$ we arrive at the familiar transformation term

$$\bar{T}^j = a_{ij} T^j$$

where the factors a_{ij} represents the elements of a transformation matrix for *contravariant vector components* between the unprimed 'old' $(r, \theta) = (x^1, x^2)$ and the primed 'new' $(x, y) = (\bar{x}^1, \bar{x}^2)$ coordinates.

3rd But these factors $\frac{\partial \bar{x}^j}{\partial x^k}$ in 6. are also the components of the basis vectors tangent to the original (unprimed) coordinate axes, expressed in the new (primed) coordinate system, e.g. the 1st column of the transformation matrix in 3. is the first tangential vector E_r w.r.t. the position vector $R(r, \theta) := (r\cos(\theta), r\sin(\theta)) = (x, y)$

$$\begin{aligned} E_r &= \frac{\partial \bar{x}^1}{\partial x^1} \cdot E_x + \frac{\partial \bar{x}^2}{\partial x^1} \cdot E_y = \cos(\theta) \cdot E_x + \sin(\theta) \cdot E_y = (\cos(\theta), \sin(\theta)) \stackrel{!}{=} \frac{\partial R}{\partial r} \\ E_\theta &= \frac{\partial \bar{x}^1}{\partial x^2} \cdot E_x + \frac{\partial \bar{x}^2}{\partial x^2} \cdot E_y = -r\sin(\theta) \cdot E_x + r\cos(\theta) \cdot E_y = (-r\sin(\theta), r\cos(\theta)) \stackrel{!}{=} \frac{\partial R}{\partial \theta} \end{aligned}$$

with the Cartesian basis $E_x := \left[\begin{smallmatrix}1\\0\end{smallmatrix}\right], E_y := \left[\begin{smallmatrix}0\\1\end{smallmatrix}\right]$, i.e. we have

$$E_r = \frac{\partial x}{\partial r} \cdot E_x + \frac{\partial y}{\partial r} \cdot E_y, \qquad E_\theta = \frac{\partial x}{\partial \theta} \cdot E_x + \frac{\partial y}{\partial \theta} \cdot E_y$$

Task:
a. Redo the discussion using MAXIMA. Calculate $E_{r=2}$ and $E_{\theta=30^\circ}$.
b. Determine the normalized polar unit vectors $\hat{E}_r$ and $\hat{E}_\theta$.[15]
c. Determine the polar line element s^2 using infinitesimal argumentation, see fig. 13.[16]

[15] We note the unit vectors with 'hat'-sign, i.e. $\hat{a} := a/|a|$.
[16] from MARSDEN, [16, p.500]

Solution:
ad **a.** and **b.** :

```
fpprintprec: 5;      /* Show only 5 significant digits in output */

x(r, theta) := r*cos(theta)$
y(r, theta) := r*sin(theta)$

U : [r, theta]$                 /* the polar coordinates .. and */
diff( x(r,theta), r);          /* the 4 elements of the Jacobean matrix */
diff( x(r,theta), theta);     /* alias the transformation matrix       */
diff( y(r,theta), r);        /* from polar (r,theta)                  */
diff( y(r,theta), theta);   /*    to Cartesian (x,y) coordinates.    */

/* Jacobean of (x,y) w.r.t. each polar variable */
define( J(r,theta) , jacobian ([x(r,theta), y(r,theta)], [r,theta]));

J(2,%pi/6);

Ex : [1,0];
Ey : [0,1];

Er     : diff( x(r,theta), r) * Ex     + diff( y(r,theta), r) * Ey;
Etheta : diff( x(r,theta), theta) * Ex + diff( y(r,theta), theta) * Ey;

load ("eigen")$   /* ad task b. */
assume(r>0)$
Eru     : trigsimp( unitvector (Er));
Ethetau : trigsimp( unitvector (Etheta));

Er2      : subst([r=2, theta=%pi/6], Er);
Etheta30 : subst([r=2, theta=%pi/6], Etheta);
```

▷ Click here to RUN the code in MAXIMA^{online}. YAMWI output:

```
define( J(r, θ) , jacobian ([x(r,θ), y(r,θ)], [r, θ]));
```

[%o8] $J\left(r,\theta\right) := \begin{pmatrix} \cos\theta & -r\sin\theta \\ \sin\theta & r\cos\theta \end{pmatrix}$

[%i9]

[%o9] $\begin{pmatrix} \frac{\sqrt{3}}{2} & -1 \\ \frac{1}{2} & \sqrt{3} \end{pmatrix}$

```
[%i10] Ex : [1,0];
```

[%o10] $[1,0]$

```
[%i11] Ey : [0,1];
```

[%o11] $[0,1]$

```
[%i12] Er : diff( x(r,θ), r)*Ex + diff( y(r,θ), r)*Ey;
```

[%o12] $[\cos\theta, \sin\theta]$

```
[%i13] Eθ : diff( x(r,θ), θ)*Ex + diff( y(r,θ), θ)*Ey;
```

[%o13] $[-r\sin\theta, r\cos\theta]$

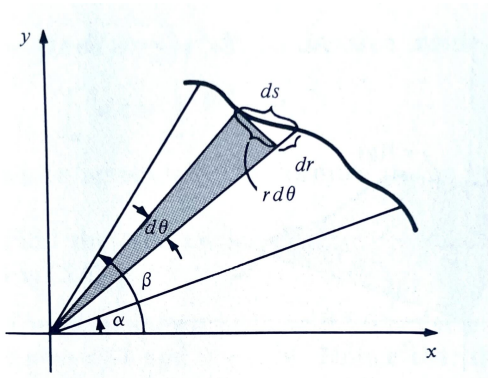


Figure 14: PYTHAGORAS' theorem in rectangular triangle $\triangle(ds, dr, r\cdot d\theta)$ says $ds^2 = dr^2 + (r\cdot d\theta)^2$, ergo we get the line element $ds = \sqrt{dr^2 + (r\cdot d\theta)^2}$ as infinitesimal length, [16, p.500].

ad **c.** [17]

> "Consider now ordinary 2D polar coordinates. Let us set up an orthonormal frame at (r, ϕ), so that its x-axis points in the r-direction and the y-axis is perpendicular to it. Now consider the polar coordinate components of an infinitesimal displacement vector $d\mathbf{v} = (dr, d\phi)$ evaluated at (r, ϕ). In the orthonormal frame itself, the coordinates of this same infinitesimal displacement vector are $d\mathbf{v} = (dr, r\cdot d\phi)$, cf. fig.14. What are the components of this vector in a global Cartesian coordinate system (x, y)? They would be $(dr\cdot\cos\phi - d\phi\cdot\sin\phi,\ dr\cdot\sin\phi + d\phi\cdot\cos\phi)$, a plain rotation!"

Let's do it in MAXIMA with package `ctensor`.

```
fpprintprec: 5;      /* Show only 5 significant digits */
postfix("^t")$ "^t"(x) := transpose(x)$  /*1 */
load(ctensor)$
cframe_flag : true$
ct_coordsys(polar)$
cmetric();
dv : [dr, r*diff(phi)]^t;          /* 2 */
trigsimp(fr);                     /* 3 */
trigsimp(fr . dv);               /* 4 */
```

▷ Click here to RUN the code.

Comment. In 1: we define local a transpose operator $..^t$ to make writing of column vectors easier. 2: defines the line element $ds^2 = 1\cdot dr^2 + r^2\cdot d\phi^2$ in polar coordinates in vector representation. We transpose this vector for reasons of mathematical correctness, although MAXIMA would allow to write $dv : [dr, r{*}d\phi]$. 3: calls 'fr' which represents the *contravariant*

[17] following the message of V. TOTH in 59309247

Cartesian coordinate components of the set of basis vectors, aligned with polar coordinates in the Euclidean plane. We can see that one basis vector in a background Cartesian frame is $(\cos\phi, \sin\phi)$ and the other is $(-\frac{1}{r}\sin\phi, \frac{1}{r}\cos\phi)$. In 4: the transformation is done, cf. 1^{st}.

YAMWI output:

```
[%i7] dv : [dr,r*dϕ];
```

$$[dr, d\phi\, r]$$

```
[%i8] trigsimp (fr);
```

$$\begin{bmatrix} \cos\varphi & -\left(\frac{\sin\varphi}{r}\right) \\ \sin\varphi & \frac{\cos\varphi}{r} \end{bmatrix}$$

```
[%i9]
```

$$\begin{bmatrix} dr\,\cos\varphi - d\phi\,\sin\varphi \\ dr\,\sin\varphi + d\phi\,\cos\varphi \end{bmatrix}$$

Exercise 9. (components of a contravariant resp. covariant vector) Do the following three exercises from the book of E. KREYSZIG [9, p.129]

DIFFERENTIALGEOMETRIE

VON

DR. ERWIN KREYSZIG

PROFESSOR AN DER TECHNISCHEN HOCHSCHULE GRAZ

Aufgaben

30.1. $a^1 = 2$, $a^2 = 1$ seien die Komponenten eines kontravarianten Vektors im Punkt P mit den Koordinaten $u^1 = 1$, $u^2 = 1$. Man bestimme die Komponenten $\bar{a}^1$, $\bar{a}^2$ dieses Vektors bezüglich der Koordinaten $\bar{u}^1 = \sqrt{(u^1)^2 + (u^2)^2}$, $\bar{u}^2 = \arctan(u^2/u^1)$.

30.2. Man bestimme die Komponenten des kovarianten Vektorfeldes $a_1 = (u^1)^2 - (u^2)^2$, $a_2 = 2u^1u^2$ bezüglich der Koordinaten $\bar{u}^1$, $\bar{u}^2$ in Aufg. 30.1.

30.3. Man bestimme die Komponenten des kontravarianten Vektorfeldes $a^1 = (u^1)^2 - (u^2)^2$, $a^2 = 2u^1u^2$ bezüglich der Koordinaten $\bar{u}^1$, $\bar{u}^2$ in Aufg. 30.1.

Lösungen der Aufgaben mit ungerader Nummer

30.1. $\bar{a}^m = a^i \dfrac{\partial \bar{u}^m}{\partial u^i} = 2\dfrac{\partial \bar{u}^m}{\partial u^1} + \dfrac{\partial \bar{u}^m}{\partial u^2}$ usw., $\bar{a}^1 = \dfrac{3}{\sqrt{2}}$, $\bar{a}^2 = -\dfrac{1}{2}$

30.3. $\bar{a}^1 = (\bar{u}^1)^2 \cos\bar{u}^2$, $\bar{a}^2 = \bar{u}^1 \sin\bar{u}^2$

5.3.3 Example: the gradient ∇f transforms *covariantly*

Imagine a square hotplate $U \subset \mathbb{R}^2$ where you can measure the corresponding temperature value $f(x, y)$ at each point (x, y) with a function $f : U \to \mathbb{R}^2$.
Then we define the gradient $grad(f) \equiv \nabla f$ of the function f by

$$grad(f) := \nabla f := (\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}) = \frac{\partial f}{\partial x} \cdot E_x + \frac{\partial f}{\partial y} \cdot E_x \tag{5.1}$$

with the Cartesian basis $E_x := \begin{bmatrix}1\\0\end{bmatrix}, E_y := \begin{bmatrix}0\\1\end{bmatrix}$.

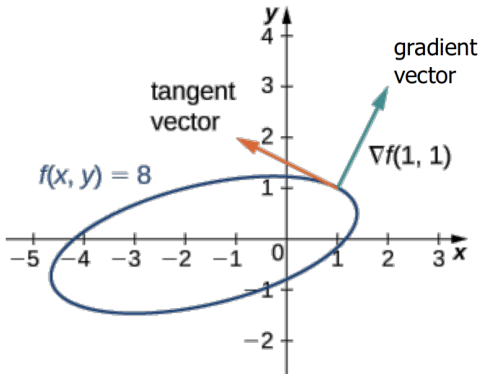


Figure 15: ◯: the graph (level surface) of $f(x, y) = 8$ for $f(x, y) = x^2 - 2xy + 5y^2 + 3x - 2y + 3$ in $\mathbb{R}^2$.
∇f: the $grad_{(1,1)} f = \nabla f(1, 1) = (3, 6)$ at point $(1, 1)$.

Exercise 10. Verify the calculation in fig. 15[18] and determine $\nabla_{(x,y)} f$, i.e. in short ∇f.
Solution: ▷ Click here to SEE the solution.

We want to show that t*he gradient vector* ∇f*, viewed as a tensor*, transforms *covariantly* from one coordinate system to another. To see, what this means, we start abstract this time and transform the gradient *from unprimed coordinates* x^1, x^2 *to primed* $\bar{x}^1, \bar{x}^2$ coordinates. If we introduce the new coordinates using the position vector $R(x^1, x^2) := (\bar{x}^1, \bar{x}^2)$, we have $\bar{x}^1 \equiv \bar{x}^1(x^1, x^2)$ and $\bar{x}^2 \equiv \bar{x}^2(x^1, x^2)$, and therefore make the dependence of $\bar{x}^1$ resp. $\bar{x}^1$ from x^1 and x^2 explicit. Now we can use the chain rule to calculate the gradient w.r.t the new coordinates. We take the same steps as in §5.3.2. Remember: the notation x^2 means the second coordinate of $\mathbf{x} = (x^1, x^2)$, _not_ the square of x!

1. because $\bar{x}^1$ depends on x^1 and x^2 etc.

$$\frac{\partial f}{\partial \bar{x}^1} = \frac{\partial f}{\partial x^1} \cdot \frac{\partial x^1}{\partial \bar{x}^1} + \frac{\partial f}{\partial x^2} \cdot \frac{\partial x^2}{\partial \bar{x}^1}$$

$$\frac{\partial f}{\partial \bar{x}^2} = \frac{\partial f}{\partial x^1} \cdot \frac{\partial x^1}{\partial \bar{x}^2} + \frac{\partial f}{\partial x^2} \cdot \frac{\partial x^2}{\partial \bar{x}^2}$$

[18] Figure from LibreTexts, Ex.14.6.4

2. write as matrix equation using *row notation* for covariant vectors

$$\left[\frac{\partial f}{\partial \bar{x}^1}, \frac{\partial f}{\partial \bar{x}^2}\right] = \left[\frac{\partial f}{\partial x^1}, \frac{\partial f}{\partial x^2}\right] \bullet \begin{bmatrix} \frac{\partial x^1}{\partial \bar{x}^1} & \frac{\partial x^1}{\partial \bar{x}^2} \\ \frac{\partial x^2}{\partial \bar{x}^1} & \frac{\partial x^2}{\partial \bar{x}^2} \end{bmatrix}$$

3. or write both equations of 1. *as a sum*

$$\frac{\partial f}{\partial \bar{x}^1} = \sum_{k=1}^{2} \frac{\partial f}{\partial x^k} \cdot \frac{\partial x^k}{\partial \bar{x}^1}, \qquad \frac{\partial f}{\partial \bar{x}^2} = \sum_{k=1}^{2} \frac{\partial f}{\partial x^k} \cdot \frac{\partial x^k}{\partial \bar{x}^2}$$

4. write both equations in 3. *as one term* with free index $j = 1..2$

$$\frac{\partial f}{\partial \bar{x}^j} = \sum_{k=1}^{2} \frac{\partial f}{\partial x^k} \cdot \frac{\partial x^k}{\partial \bar{x}^j}$$

5. write even more compact using EINSTEIN summation convention

$$\frac{\partial f}{\partial \bar{x}^j} = \frac{\partial f}{\partial x^k} \cdot \frac{\partial x^k}{\partial \bar{x}^j}$$

6. write the transformation formula 5. as abstract pattern using $\bar{T}_j := \frac{\partial f}{\partial \bar{x}^j}$ and $T_j := \frac{\partial f}{\partial x^j}$

$$\bar{T}_j = \frac{\partial x^k}{\partial \bar{x}^j} \cdot T_j$$

• Note: in 6. are the coordinate transformation equations for the components of a covariant vector, where T_j are the components of the tensor (vector) in the original coordinate system x^j and $\bar{T}_j$ are the components of the same tensor (vector) expressed in the new coordinate system $\bar{x}^j$.

• Note also, in this case the transformation matrix $(\frac{\partial x^k}{\partial \bar{x}^j})$ is the *inverse* of the transformation matrix of the line element in $(\frac{\partial \bar{x}^j}{\partial x^k})$ in §5.3.2-6.! Hence in this case the weighting factors are the components of the contravariant basis vectors, which means that the components of the gradient vector transform as covariant components.

◦ Therefore many authors define the covariant components of a vector as components that transform according to equation 6.

Task: Let $f(x, y) = x^2 - 2xy + 5y^2 + 3x - 2y + 3$ in $\mathbb{R}^2$ be the function to be used. Redo the discussion using MAXIMA for the transformations $(x, y) \mapsto (r, \theta)$ resp. $(x, y) \leftarrow (r, \theta)$ and calculate the gradient of f in both coordinate systems.

Solution:

1st To calculate gradients and Jacobians of f, we could do, e.g., this:

```
f(x,y) := x^2-2*x*y+5*y^2+3*x-2*y+3;

(" grad of f in Cartesian coords ")$
E : [x, y];     /* Cartesian coordinates */
gradf : [diff( f(x,y), E[1]),  diff( f(x,y), E[2])];
/* or equivalent using makelist() */
gradf :  makelist( diff( f(x,y), E[i]),  i,1,2);

(" Jacobian  of f in polar coords ")$
/*-- contributed by Viktor Toth --*/
U : [r, th];                  /* polar coords:  r, theta */
assume( r > 0 );

J1 : apply(matrix, makelist( makelist(                              /*1:*/
          diff( ev(U[j], r=sqrt(x^2+y^2), th=atan2(y,x)), E[i]),   /*2:*/
                        i,1,2),   j,1,2));                        /*3:*/

J2 : apply(matrix,makelist(makelist(
          diff( ev( E[j], x=r*cos(th), y=r*sin(th)),  U[i]),   /*4:*/
                i,1,2),   j,1,2));

J1.J2,  x^2+y^2=r^2, sin(th)=y/r, cos(th)=x/r, factor, eval; /*5:*/

/* proving that J1 and J2 are inverses */
 define( funmake(f1, [r,th]), ev( f(x,y), x=r*cos(th), y=r*sin(th)));/ *6:*/

/* proving that gradf1 is the transformed gradf */
gradf1: makelist( diff( f1(r,th), U[i]),i,1,2);
(gradf . J2)[1] - gradf1, x=r*cos(th), y=r*sin(th), trigsimp, radcan;/*7:*/
```

Comment. In 1: we `apply` the `matrix` constructor on a double list (3:), which holds the individual elements of the JACOBI matrix of f. Each entry is the the `ev()`aluation of the polar coordinate U_j (where r and Θ are substituted with $x's$ and $y's$!) and the result `diff()`erentated w.r.t the Cartesian coordinate E_i. Sigh. This is a clever construct! In 4: we do the same procedure: substitute x and y in E_j by r and Θ and then differentiate the result w.r.t the ith polar coordinate U_i. 5: calculates the matrix product of the two jacobian's using three substitutions, factors and evaluates the result. 6: invokes `fun`(ction)`make` to rewrite function f using polar coordinates. In line 7: we check the polar transformed gradient of function f and the new calculated gradient $gradf1$ for equality, i.e. if their difference is zero.

▷ Click here to RUN the code in MAXIMA^{online}.

YAMWI output:

```
[%i9]  J1 : apply(matrix, makelist( makelist(
                 diff( ev(U[j], r=sqrt(x^2+y^2), th=atan2(y,x)), E[i]),
                                  i,1,2),   j,1,2));
```

[%o9]

$$\begin{bmatrix} \frac{x}{\sqrt{y^2+x^2}} & \frac{y}{\sqrt{y^2+x^2}} \\ -\left(\frac{1}{2x}\right) & \frac{1}{2x} \end{bmatrix}$$

```
[%i10] J2 : apply(matrix,makelist(makelist(
                 diff( ev( E[j], x=r*cos(th), y=r*sin(th)),  U[i]),
                       i,1,2),   j,1,2));
```

[%o10]

$$\begin{bmatrix} \cos th & -(r \sin th) \\ \sin th & r \cos th \end{bmatrix}$$

```
[%i11] J1.J2,  x^2+y^2=r^2, sin(th)=y/r, cos(th)=x/r, factor, eval;
```

[%o11]

$$\begin{bmatrix} 1 & 0 \\ \frac{y-x}{2rx} & \frac{y+x}{2x} \end{bmatrix}$$

```
[%i12] /* proving that J1 and J2 are inverses */
       define( funmake(f1, [r,th]), ev(f(x,y), x=r*cos(th), y=r*sin(th)));
```

[%o12]

$$f_1(r, th) := 5\,r^2\,(\sin th)^2 - 2\,r^2\,\cos th\,\sin th - 2\,r\,\sin th + r^2\,(\cos th)^2 + 3\,r\,\cos th + 3$$

```
[%i13] /* proving that gradf1 is the transformed gradf */
       gradf1: makelist( diff( f1(r,th), U[i]),i,1,2);
```

[%o13]

$$\left[10\,r\,(\sin th)^2 - 4\,r\,\cos th\,\sin th - 2\,\sin th + 2\,r\,(\cos th)^2 + 3\,\cos th, 2\,r^2\,(\sin th)^2 + 8\,r^2\,\cos th\,\sin th - 3\,r\,\sin th - 2\,r^2\,(\cos th)^2 - 2\,r\,\cos th\right]$$

```
[%i14] (gradf . J2)[1] - gradf1, x=r*cos(th), y=r*sin(th), trigsimp, radcan;
```

[%o14]

$$[0, 0]$$

2nd Note that we used explicit formulas in step **1**$^{\mathbf{st}}$ but we could have used the built-in 'jacobian' function to calculate the transformation matrices:

```
/*-- contributed by Viktor Toth --*/
E : [x, y];          /* Cartesian coordinates: x,y   */
U : [r, th];         /* polar coordinates:  r, theta */
assume( r > 0 );

J1 : jacobian( ev(U, r=sqrt(x^2+y^2), th=atan2(y,x)), E);          /*1*/
J2 : jacobian( ev(E, x=r*cos(th),     y =r*sin(th)),  U);         /*2*/
```

Comment. In 1: we construct the transformation matrix $J1 : \overset{polar}{U} \mapsto \overset{Cartesian}{E}$ using the `jacobian` function of MAXIMA. In 2: we reverse the process:

$$J2 : \overset{cartesian}{E} \mapsto \overset{polar}{U}$$

▷ Click here to run the code in MAXIMA^{online}.

YAMWI output:

```
[%i4] J1 : jacobian( ev(U, r=sqrt(x^2+y^2), th=atan2(y,x)), E);
```

[%o4]

$$\begin{bmatrix} \frac{x}{\sqrt{y^2+x^2}} & \frac{y}{\sqrt{y^2+x^2}} \\ -\left(\frac{1}{2x}\right) & \frac{1}{2x} \end{bmatrix}$$

[%i5]

[%o5]

$$\begin{bmatrix} \cos th & -(r\,\sin th) \\ \sin th & r\,\cos th \end{bmatrix}$$

3rd As a surplus we demonstrate how to use the function `cograd()` of package `ctensor` to calculate the gradient of f.

```
/*-- contributed by Viktor Toth --*/
load(ctensor);
ct_coordsys(cartesian2d);
f(x,y) := x^2-2*x*y+5*y^2+3*x-2*y+3;
gradf : [0,0]; /* preinitialize gradf, so it's treated as a proper list */
cograd( f(x,y), gradf );
gradf;
subst([x=1,y=1], gradf);
```

▷ Info about `cograd`.
▷ Click here to run the code in MAXIMA^{online}.

YAMWI output:

```
[%i3] f(x,y) := x^2-2*x*y+5*y^2+3*x-2*y+3;
```

[%o3] $f(x,y) := 5\,y^2 - 2\,x\,y - 2\,y + x^2 + 3\,x + 3$

```
[%i4] gradf : [0,0];
```

[%o4] $[0, 0]$

```
[%i5] /* preinitialize gradf, so it's treated as a proper list */
      cograd( f(x,y), gradf );
```

[%o5] **done**

```
[%i6] gradf;
```

[%o6] $[-(2\,y) + 2\,x + 3, 10\,y - 2\,x - 2]$

```
[%i7] subst([x=1, y=1], gradf);
```

[%o7] $[3, 6]$

Exercise 11. (the velocity vector of a curve in polar coordinates)
Let $y = x^2$ alias $C(t) := (t, t^2)$ be the curve ('parabola') in the Cartesian plane $\mathbb{R}^2$.

a. Show, that $C(t)$ has the equation $c(\Theta) = (\frac{\sin(\Theta)}{\cos(\Theta)^2}, \Theta)$ in polar coordinates.
Remember, both curves $C(t)$ and $c(\Theta)$ parameterize the same set ('manifold').

b. Verify: the Cartesian point $(x = 1, y = 1)$ is the same point as $(r = \sqrt{2}, \theta = \frac{\pi}{4})$ in polar coordinates.

c. Calculate and verify in the figure: the velocity vector v at $(1,1)|_{cartesian}$ is $(1,2)|_{cartesian}$, i. e. in the Cartesian basis $E_x := \left[\begin{smallmatrix}1\\0\end{smallmatrix}\right], E_y := \left[\begin{smallmatrix}0\\1\end{smallmatrix}\right]$ we have $P = 1 \cdot E_x + 1 \cdot E_y$ and $v = 1 \cdot E_x + 2 \cdot E_y$.

What are the polar coordinates of v? Check your result at the picture.

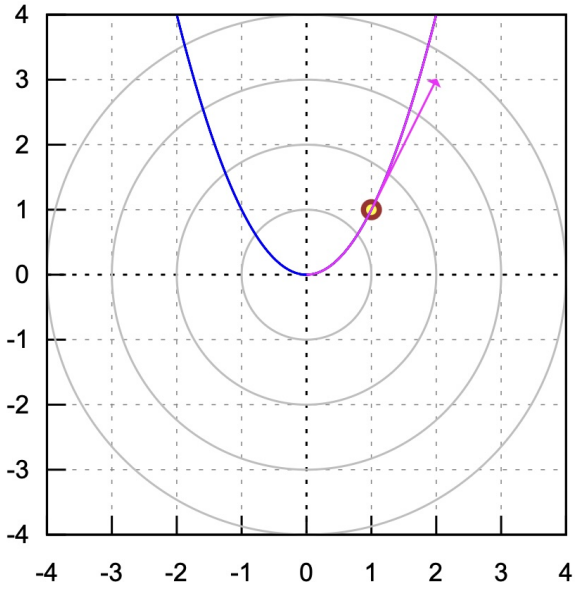


Figure 16:
⊙: the polar coordinates aligned in the Euclidean plane.
●: the Cartesian point $(1,1) \overset{polar}{\mapsto} (\sqrt{2}, \frac{\pi}{4})$ (brown ring)
⌣: the Cartesian curve $y = x^2$ on $[0, 2]$.
⌣: the polar curve $c(\Theta) = (\frac{\sin(\Theta)}{\cos(\Theta)^2}, \Theta)$ on $[0, 2\pi]$.

6 KRONECKER delta tensor δ_i^j

In Linear Algebra, the KRONECKER delta (named after Leopold KRONECKER) is a *function* of two variables ('indices'), usually non-negative integers. This function has value 1 if the inputs are equal, and 0 otherwise. It is _*not*_ a tensor. In contrast, the KRONECKER delta *tensor* of the MAXIMA package `itensor` is a true tensor.

6.1 Definition: δ_i^j

The KRONECKER delta *tensor* of Tensor Calculus is defined by

$$\delta_i^j := \begin{cases} 0 & \text{if } i \neq j, \\ 1 & \text{if } i = j. \end{cases} \tag{6.1}$$

$$= \texttt{kdelta}([i],[j]) \quad \text{name in itensor package} \tag{6.2}$$

Remark.

1. `kdelta(L1, L2)` is the *generalized Kronecker delta tensor* defined in the `itensor` package with $L1$ the list of covariant indices and $L2$ the list of contravariant indices. In particular δ_i^j *is a tensor, but* δ_{ij} *is not.* For a proof cf. [7, 82], "which shows that the values of the KRONECKER delta symbol in different coordinate systems relate by the same tensor rule."

2. The KRONECKER delta *function* of Linear Algebra is defined by

$$\delta_{ij} := \begin{cases} 0 & \text{if } i \neq j, \\ 1 & \text{if } i = j. \end{cases} \tag{6.3}$$

$$= \texttt{kron\_delta}(i,j) \quad \text{in Linear Algebra package} \tag{6.4}$$

   In Maxima, this function δ_{ij} is named `kron_delta`.
   This δ_{ij} *is not a tensor and should not be used in the context of Tensor Calculus.*

3. To sum up: We have two versions of the KRONECKER δ symbol in MAXIMA, a function and a tensor:

LEXICON	*Math*	MAXIMA
KRONECKER δ function	$\delta_{i,j}$	`kron_delta(i,j)`
KRONECKER δ tensor	δ_a^b	`kdelta([a],[b])`

   • In the following we will exclusively use `kdelta([a],[b])` because it acts also as the generalized version of the KRONECKER δ tensor.

4. δ_i^j is the identity transformation that transforms a vector into itself. That is a coordinate-free statement: the components of the KRONECKER delta remain the same in every coordinate system. The KRONECKER delta δ_i^j always has the same number of upstairs and downstairs indices, and so respects the transformation rules.

Example 7. (some checks of δ_i^j vs. δ_{ij})

```
("delta is the Kronecker delta function of Linear Algebra")$
delta(i,j) := kron_delta(i,j)$

delta(2,2);
delta(1,2);

("kdelta is the Kronecker delta with special contraction properties")$
load(itensor)$
dim : 3;
ishow(kdelta([a],[b]))$
ishow(kdelta([2],[2]))$
ishow(kdelta([1],[2]))$
ishow(kdelta([a,b],[c,d]))$
```

▷ Click here to RUN the code.

YAMWI output:

```
[%i8]  ishow(kdelta([2],[2]))$
[%o8]                                  1
[%i9]  ishow(kdelta([1],[2]))$
[%o9]                                  0
[%i10] ishow(kdelta([a,b],[c,d]))$
[%o10]                  c       d         d       c
                 kdelta  kdelta  - kdelta  kdelta
                        a       b         a       b
```

Comment. In Line (%i8) and (%i9) we test the tensor `kdelta` of `itensor` package on the input pair $(2,2)$ and on the pair $(1,2)$. In (%i10) we calculate δ_{ab}^{cd} and verify the relation $\delta_{ab}^{cd} = \delta_a^c \cdot \delta_b^d - \delta_a^d \cdot \delta_b^c$ in (%o10), i.e. the generalized KRONECKER delta tensor δ_{ab}^{cd} is reduced to the 'original single' delta tensor of definition (6.2).

Remark.

1. Considered as a type (1,1) tensor, the Kronecker tensor is written as δ_i^j , with one covariant index i and contravariant index j. We say: δ_i^j is a rank-2 mixed tensor.
2. SAGEMATH distinguishes also between a function and a tensor, both noted by δ_{ij}.

- For symbolic calculations, SAGEMATH use the `kronecker_delta` function:

```
print(kronecker_delta(1, 2))  # Output: 0
```

- For calculations in tensor modules `KroneckerDelta` is used, see Sage:

```
from sage.tensor.modules.comp import KroneckerDelta
V = VectorSpace(QQ,2);  d = KroneckerDelta(QQ, V.basis()) ;
d[:]; [1 0]
      [0 1]
```

Example 8. (some uses of δ_i^j)
For concreteness let's assume 3D space; any index runs over the three dimensions. We use Cartesian coordinates x, y, z or more general x^1, x^2, x^3. The indices varies through $1, 2, 3$.

a. What is $\frac{\partial x^i}{\partial x^j}$?

b. What gives $A_i \delta_j^i = A_j$?

c. What means $A_j^i B_k^j = \delta_k^i$?

d. $\delta_i^j A_i^j =$?

e. $\delta_i^i =$?

Answer:

a: δ_i^j, because different independents x^i have no dependence on each other.

b: $A_i \delta_j^i \overset{Einstein}{=} \sum_{i=1}^3 A_i \delta_j^i$, where $j = 1, 2, 3$ is free.
So we have 3 equations incorporated in this single tensorial term:
$A_1\delta_1^1 + A_2\delta_1^2 + A_3\delta_1^3 = A_1$
$A_1\delta_1^2 + A_2\delta_2^2 + A_3\delta_2^3 = A_2$
$A_1\delta_1^3 + A_2\delta_2^3 + A_3\delta_3^3 = A_3$

c: A and B are inverse matrices, i.e. $A.B = E$.

d: A_i^i, which is a 'contraction' i.e. a renaming to make a common lower and upper index which we then sum over.

e: 3, because we assumed 3 as dimension of the space, therefore we have:
$\delta_i^i \overset{Einstein}{=} \sum_{i=1}^3 \delta_i^i = \delta_1^1 + \delta_2^2 + \delta_3^3 = 1 + 1 + 1 = 3$

Example 9. (δ_i^j is a true tensor)
To *prove*: δ_i^j with one upper and one lower index is a rank-2 mixed tensor.

Proof: From COLLIER [5, p.143].
Let $(x^1, .., x^n)$ be the original 'old' coordinate system and $(y^1, .., y^n)$ the primed 'new' one. Using index notation we have to verify the tensor property for the transform of the partial derivatives, i.e. δ_i^j should transform according to

$$\bar{\delta}_k^\ell = \frac{\partial y^\ell}{\partial x^j} \cdot \frac{\partial x^i}{\partial y^k} \cdot \delta_i^j$$

We have

$$\begin{aligned} \bar{\delta}_k^\ell &= \frac{\partial y^\ell}{\partial x^j} \cdot \frac{\partial x^i}{\partial y^k} \cdot \delta_i^j \\ &\overset{1}{=} \frac{\partial y^\ell}{\partial x^j} \cdot \frac{\partial x^j}{\partial y^k} \\ &\overset{2}{=} \frac{\partial y^\ell}{\partial y^k} \\ &= \delta_k^\ell \end{aligned}$$

1: the right-hand side KRONECKER delta relabels the i index as j.
2: the primed new coordinates are independent of each other.

Comment. In this proof we realize that the KRONECKER delta tensor δ_k^ℓ is used as a relabeling of indices and therefore simplifying tensorial expressions.

6.2 Example: The case of the Scalar product of Vectors

The inner product (aka scalar product) of vectors in Linear Algebra is often written using the symbol δ_{ij} (as function, not as tensor!) as

$$\mathbf{a} \cdot \mathbf{b} = \sum_{i=1}^{n} \sum_{j=1}^{n} a_i \delta_{ij} b_j = \sum_{i=1}^{n} a_i b_i$$

where i and j take the values $1, 2, ..., n$ and the vectors $\mathbf{a}$ and $\mathbf{b}$ are defined as arbitrary n-tuples $\mathbf{a} = (a_1, a_2, \ldots, a_n)$ and $\mathbf{b} = (b_1, b_2, ..., b_n)$ in $\mathbb{R}^n$.
We see how in the above equation the values of the KRONECKER delta function δ_i^j reduce the double summation over i and j to a single summation over i only ...

▷ But this is an *abuse of the* EINSTEIN *summation convention:* A summation in a tensor expression is only implicit done, when an index appears twice, once as a *subscript* ('lower index') and once as a *superscript* ('upper index'). This is not the case here!

Task: define a correct tensorial scalar product using e.g. MAXIMA's tensor packages.
Remember, that the scalar product in Linear Algebra is defined by the GRAMmatrix g, i.e. the metric tensor bythe tensorial pattern

$$g_{ij} A^i B^j \tag{6.5}$$

(instead of $\delta_{ij} a_i b_j$), where A and B are column vectors! Herein the case $g_{ij} = \delta_{ij}$ is only correct, if the corresponding basis is orthonormal in the Euclidean space.

1st definition of the inner product using plain MAXIMA.

```
sp(g,A,B) :=  sum(sum( g[i][j]*A[i]*B[j], j,1,length(g)),i,1,length(g))[1];
g : [[1,0,0],[0,1,0],[0,0,1]];
A : transpose([a1,a2,a3]);
B : transpose([b1,b2,b3]);
sp(g,A,B);
```

▷ Click here to RUN the code.
YAMWI output:

```
[%i2] g : [[1,0,0],[0,1,0],[0,0,1]];
[%o2]                    [[1,0,0],[0,1,0],[0,0,1]]
[%i3] A : transpose([a1,a2,a3]);
[%o3]                    [ a1 ]
                         [ a2 ]
                         [ a3 ]
[%i4] B : transpose([b1,b2,b3]);
[%o4]                    [ b1 ]
                         [ b2 ]
                         [ b3 ]
[%i5] sp(g,A,B);
[%o5]                    a3 b3 + a2 b2 + a1 b1
```

Comment. We implement the term $g_{ij}A^iB^j$ as `g[i][j]*A[i]*B[j]`, defining the metric g "by hand", represented as a 3-by-3-matrix and use the size[19] of the metric g tensor explicitly in the summation. So the definition is independent of the dimension of the space. The double sum resembles the Linear Algebra definition for the scalar product in Cartesian vector space $\mathbb{R}^3$ from the beginning. Function `sp()` expects as arguments the metric g and two column vectors - therefore we transpose the lists for A and B.

2nd definition of the inner product using `ctensor`.

We will use `ctensor`, which works only after a specific coordinate system is choosen. Then we may do ordinary matrix-vector multiplication a la "$\_\cdot\square\cdot \mid \rightsquigarrow \_\cdot \mid \rightsquigarrow \bullet$" to calculate the inner product:

```
load(ctensor);
ct_coordsys(cartesian3d);
A : [a1,a2,a3];
B : [b1,b2,b3];
A.lg.transpose(B);
```

Comment. by V. TOTH: 'the catch: *strictly for convenience,* the metric $g = lg$ is represented using a matrix, which would imply a mixed-rank tensor, which the metric most specifically isn't. But, well, let's go along with it, change one row index into a column index, which means changing a column index into a row index, breaking the representational rule that rows and columns correspond to covariant vs. contravariant quantities, just so that we can then do this' for the inner product: $A \cdot lg \cdot transpose(B)$, which realizes $g_{ij}A^iB^j$.

▷ Click here to RUN the code.

YAMWI output:

```
[%i3] A:[a1,a2,a3];
[%o3]                                  [a1, a2, a3]
[%i4] B:[b1,b2,b3];
[%o4]                                  [b1, b2, b3]
[%i5] A.lg.transpose(B);
[%o5]                          a3 b3 + a2 b2 + a1 b1
```

Exercise 12. Write a MAXIMA function `sp()` based on expression (%i5).

[19] the `length(g)` of a matrix is the number of the rows

3rd definition of the inner product using `i/c/tensor`.

The following code and the insightful inline comments are contributed by V. TOTH[20]. Let's listen to his explanations:

```
load(itensor);
imetric(g);                                                    /* 1: */
     /*  I do _not_ want to carry out the inner product using abstract
      *  indices. I want the _unevaluated_ inner product expression.
      *  Moreover, I want it to be in a form that ‘ic_convert‘ can
      *  process that is, an assignment: */
ishow(c = g([i,j],[]) * A([],[i]) * B([],[j]) )$               /* 2: */
sp : ic_convert(%);                                            /* 3: */
     /* Now I can move to ‘ctensor‘ and define a coordinate reference
      * frame explicitly, e.g.  specifying our coordinate basis,
      * our dimensionality, our metric: */
load(ctensor);
dim : 3;
ct_coords : [x,y,z];
lg : ident(3);
     /* And _now_ we can do this: */
A : [a1,a2,a3];
B : [b1,b2,b3];
sp, eval;
```

Comment. In 1: the variable `imetric` is bound to the metric g, assigned by the `imetric(g)` command. We define the term $g_{ij}A^iB^j$ in 2: as `g([i,j],[]) * A([],[i]) * B([],[j])`. `ishow` displays this inner product expression as correct indexed object, having their covariant indices as subscripts and contravariant indices as superscripts. In 3: the variable '`sp`' now contains the indexed expression for the inner product in value 'c', converted by the function `ic_convert` to an object, which can be further processed by `ctensor`. Remember: the system variable % catch the last expression, i.e. the equation in line 2:. Note: the variable c in 2: is a scalar value (number), which is in accordance with the real valued scalar product feature $sp : \mathbb{R}^3 \times \mathbb{R}^3 \to \mathbb{R}^1$.

▷ Click here to RUN the code.

YAMWI output:

[20] following a Socratic email lesson with him about the construction of the inner product with the itensor package. I owe Viktor TOTH important and insightful advice here, as always. So I share his expert knowledge w.r.t. the tensor packages with great respect.

```
[%i5] sp : ic_convert(c=%);
```

[%o5] $c : \mathrm{sum}\left(\mathrm{sum}\left(A_i \, \mathrm{lg}_{i,j} \, B_j, i, 1, \mathrm{dim}\right), j, 1, \mathrm{dim}\right)$

```
[%i6] dim : 3;
```

[%o6] 3

```
[%i7] ct_coords : [x,y,z];
```

[%o7] $[x, y, z]$

```
[%i8] lg : ident(3);
```

[%o8] $\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$

```
[%i9] A : [a1,a2,a3];
```

[%o9] $[a_1, a_2, a_3]$

```
[%i10] B : [b1,b2,b3];
```

[%o10] $[b_1, b_2, b_3]$

```
[%i11] sp, eval;
```

[%o11] $a_3 \, b_3 + a_2 \, b_2 + a_1 \, b_1$

4th Here is an abstraction of the above procedure to a function:

```
/*--  contributed by V. Toth  --*/
     /* Or maybe you want a functional form of  'sp'.
      * Let's take a step back to how 'sp' is defined
      * and do instead this: */
load(ctensor);
load(itensor);
imetric(g);
ct_coordsys( cartesian3d );   /* sets dim=3, ct_coords[..] and lg */
ishow( c = g([i,j],[]) * A([],[i]) * B([],[j]))$

define( sp(A,B), rhs( ic_convert(%)) );
sp( [a1,a2,a3], [b1,b2,b3] );
```

▷ Click here to RUN the code.
YAMWI output:

```
[%i6] define( sp(A,B), rhs( ic_convert(c=%)) );
```

```
[%o6] sp(A, B) := sum(sum(A  B  lg    , i, 1,
                          i  j   i, j
dim), j, 1, dim)
```

```
[%i7] sp([a1,a2,a3],[b1,b2,b3]);
```

```
[%o7] a3 b3 + a2 b2 + a1 b1
```

- To recap:

INNER PRODUCT sp()	*Tensor Math*	realization in `i/c/tensor` syntax
$\mathbf{1}^{st}$:	$g_{ij}A^iB^j$	`g([i][j])*A[i]*B[j]`
$\mathbf{2}^{nd}$:		`A . g . transpose(B)`
$\mathbf{3}^{rd}$:		`g([i,j],[]) * A([],[i]) * B([],[j])`

Only the $\mathbf{3}^{rd}$ version is the fully correct realization of the inner product in the tensorial setting of the packages `c/itensor`.

♡ We end this section with

$$\delta_i^i \equiv \sum_{i=1}^{3} \delta_i^i \qquad \text{in } \mathbb{R}^3$$

$$\downarrow \textit{ contract}$$

$$\downarrow \textit{ i.e. sum}$$

$$3$$

the EINSTEIN summation convention applied. ♡

7 permutation symbol ε

The *permutation symbol* (alias LEVI-CIVITA symbol) is a multi-index function sometimes also called the *alternating tensor* or the *signature*. There are several common notations for the symbol: some authors uses the Greek epsilon character ϵ_{ijk}, others uses the curly variant ε_{ijk} or the Latin lower case e_{ijk}. We will use the latter, because we will follow GRINFELD and reserve the Greek epsilon notation for the LEVI-CIVITA symbol, cf. [7, p.134]. The permutation symbol is heavily used to define advanced tensorial objects such as the determinant, the curl or the cross product of vectors, cf. [46].

7.1 Definition

In two dimensions, the *permutation symbol* e_{ij} is defined by:

$$e_{ij} := \begin{cases} +1 & \text{if } (i,j) = (1,2) \\ -1 & \text{if } (i,j) = (2,1) \\ 0 & \text{if } i = j \end{cases} \tag{7.1}$$

- In three dimensions, the permutation symbol is defined by:

$$e_{ijk} := \begin{cases} +1 & \text{if } (i,j,k) \text{ is } (1,2,3), (2,3,1), \text{ or } (3,1,2), \\ -1 & \text{if } (i,j,k) \text{ is } (3,2,1), (1,3,2), \text{ or } (2,1,3), \\ 0 & \text{if } i = j, \text{ or } j = k, \text{ or } k = i \end{cases}$$

- In four dimensions (and also in 2D or 3D), the permutation symbol is defined by:

$$e_{ijkl} := \begin{cases} +1 & \text{if } (i,j,k,l) \text{ is an even permutation of } (1,2,3,4) \\ -1 & \text{if } (i,j,k,l) \text{ is an odd permutation of } (1,2,3,4) \\ 0 & \text{otherwise} \end{cases}$$

- In arbitrary dimension the permutation (or Levi-Civita) symbol $e_{ijk\ldots}$ is defined in the `itensor` package of MAXIMA by the function `levi_civita()`

```
load(itensor);
levi_civita(L);
```

which yields 1 if the list L consists of an even permutation of integers, -1 if it consists of an odd permutation, and 0 if some indices in L are repeated.

Remark.

LEXICON	*Math*	MAXIMA	`itensor`
permutation symbol:	$e_{ijk\ldots}$	`eijk` or `Eijk`	`levi_civita()`

7.2 Examples

To get a feeling of the construction of the levi-civita symbol $e_{ijk...}$ we discuss some special cases.

7.2.1 ... the two dimensional permutation symbol `eij`

```
/* permutation 'tensor' as function Eij */
Eij(i,j) := j-i;
Eij(1,2);
Eij(1,1);

/* permutation 'tensor' as table eij */
eij[i,j] := j-i ;
genmatrix (eij, 2, 2);
eij[1,2];     /*-- result on index pair (1,2) */
```

YAMWI output:

```
[%i1] Eij(i,j) := j-i;
[%o1]                    Eij(i,j) := j - i
[%i2] Eij(1,2);
[%o2]                              1
[%i3] Eij(1,1);
[%o3]                              0
[%i4] eij:zeromatrix(2,2)$
[%i5] for i:1 thru 2 do for j:1 thru 2 do eij[i][j]:j-i ;
[%o5]                            done
[%i6] eij;
[%o6]                          [  0   1 ]
                               [ - 1  0 ]
[%i7] eij[1,2];
[%o7]                              1
```

▷ Click here to run the script.

Exercise 13. Define function `Eij(i,j)` using the math definition for e_{ij} from (7.1).

7.2.2 ... the three dimensional permutation symbol `eijk`

We use the 1st bullet part of §7.1 for an explicit definition and check the first test also with `itensor`'s levi-civita symbol.

```
/* 4-dim permutation symbol as function, i.e.
 * eijk is 1 if (i,j,k) is an even permutation of (1,2,3),
 *        -1 if (i,j,k) is an odd permutation of (1,2,3),
 *         0 if any index is repeated. */
```

```
Eijk(i,j,k) :=
    if      is([i,j,k]=[1,2,3]) or is([i,j,k]=[2,3,1])
                                or is([i,j,k]=[3,1,2]) then +1
    elseif  is([i,j,k]=[3,2,1]) or is([i,j,k]=[1,3,2])
                                or is([i,j,k]=[2,1,3]) then -1
    elseif  is(i=j) or is(j=k) or is(k=i) then 0 $

Eijk(1,2,3);
Eijk(2,1,3);
Eijk(2,2,1);

load(itensor);
ishow(levi_civita ([1,2,3]));
```

▷ Click here to run the script.
YAMWI output:

```
Eijk(i,j,k) :=
    if      is([i,j,k]=[1,2,3]) or is([i,j,k]=[2,3,1]) or is([i,j,k]=[3,1,2]) then +1
    elseif  is([i,j,k]=[3,2,1]) or is([i,j,k]=[1,3,2]) or is([i,j,k]=[2,1,3]) then -1
    elseif  is(i=j) or is(j=k) or is(k=i) then 0 $
[%i2] Eijk(1,2,3);
[%o2]                                    1
[%i3] Eijk(2,1,3);
[%o3]                                   -1
[%i4] Eijk(2,2,1);
[%o4]                                    0
[%i5] load(itensor)$
[%i6] ishow(levi_civita ([1,2,3]));
[%t6]                                    1
```

◦ Now a version with table access using a permutation term:

```
load(ctensor);
dim:3;
for i thru dim do
      for j thru dim do
            for k thru dim do   eijk[i,j,k] : (j-i)*(k-i)*(k-j)/2;

eijk[1,2,1];
eijk[1,2,3] + eijk[2,1,3];
eijk[1,2,3] * eijk[2,3,1];

cdisplay(eijk);
```

Comment. We use `ctensor`, because it is not possible to use `genmatrix()`, which works only for 2-dim matrices and the use of `array()` seems to cumbersome, see §5.2.

YAMWI output:

```
[%i4] eijk [1,2,1];

[%o4]                                0
[%i5] eijk[1,2,3] + eijk[2,1,3];

[%o5]                                0
[%i6] eijk[1,2,3] * eijk[2,3,1];

[%o6]                                1
[%i7] cdisplay(eijk);

[]   eijk_1 = matrix([0,0,0],[0,0,1],[0,-1,0])
[]   eijk_2 = matrix([0,0,-1],[0,0,0],[1,0,0])
[]   eijk_3 = matrix([0,1,0],[-1,0,0],[0,0,0])
```

▷ Click here to run the script.

Remark. The MAXIMA output of tensor `eijk` is verified by this visualization:

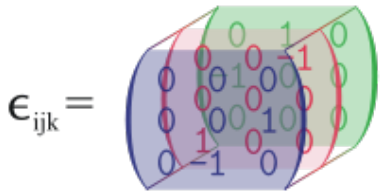


Figure 17: 3D image of `eijk` with 3 layered matrix components, [46].

Exercise 14. Define function `Eijk(i,j,k)` using the defining term `(j-i)*(k-i)*(k-j)/2`. Argue for the correctness of the formula and test it on some examples e.g. `eijk(3,2,1)=-1`.

Exercise 15. Here is a skeleton to define `eijk` explicit using only the 6 values that are not equal zero.

```
eijk = zero(3,3,3)
  eijk[1,2,3] =  1
  eijk[2,3,1] =  1
  eijk[3,1,2] =  1
  eijk[3,2,1] = -1
  eijk[1,3,2] = -1
  eijk[2,1,3] = -1
eijk   -- show the tensor
```

Explain. Realize the construction in MAXIMA.

7.2.3 ... the 4D permutation symbol `eijkl`

In what follows we will not use more than the 4 dimensional version of the permutation symbol:

$$e_{ijkl} = \left[\begin{bmatrix} 0&0&0&0\\0&0&0&0\\0&0&0&0\\0&0&0&0 \end{bmatrix} \begin{bmatrix} 0&0&0&0\\0&0&0&0\\0&0&0&1\\0&0&-1&0 \end{bmatrix} \begin{bmatrix} 0&0&0&0\\0&0&0&-1\\0&0&0&0\\0&1&0&0 \end{bmatrix} \begin{bmatrix} 0&0&0&0\\0&0&1&0\\0&-1&0&0\\0&0&0&0 \end{bmatrix}\right.$$
$$\begin{bmatrix} 0&0&0&0\\0&0&0&0\\0&0&0&-1\\0&0&1&0 \end{bmatrix} \begin{bmatrix} 0&0&0&0\\0&0&0&0\\0&0&0&0\\0&0&0&0 \end{bmatrix} \begin{bmatrix} 0&0&0&1\\0&0&0&0\\0&0&0&0\\-1&0&0&0 \end{bmatrix} \begin{bmatrix} 0&0&-1&0\\0&0&0&0\\1&0&0&0\\0&0&0&0 \end{bmatrix}$$
$$\begin{bmatrix} 0&0&0&0\\0&0&0&1\\0&0&0&0\\0&-1&0&0 \end{bmatrix} \begin{bmatrix} 0&0&0&-1\\0&0&0&0\\0&0&0&0\\1&0&0&0 \end{bmatrix} \begin{bmatrix} 0&0&0&0\\0&0&0&0\\0&0&0&0\\0&0&0&0 \end{bmatrix} \begin{bmatrix} 0&1&0&0\\-1&0&0&0\\0&0&0&0\\0&0&0&0 \end{bmatrix}$$
$$\left.\begin{bmatrix} 0&0&0&0\\0&0&-1&0\\0&1&0&0\\0&0&0&0 \end{bmatrix} \begin{bmatrix} 0&0&1&0\\0&0&0&0\\-1&0&0&0\\0&0&0&0 \end{bmatrix} \begin{bmatrix} 0&-1&0&0\\1&0&0&0\\0&0&0&0\\0&0&0&0 \end{bmatrix} \begin{bmatrix} 0&0&0&0\\0&0&0&0\\0&0&0&0\\0&0&0&0 \end{bmatrix}\right]$$

Task:

a. Implement the 4D permutation symbol `eijkl` as table using the formula

$$\text{eijk}\ell[i,j,k,l] = (i-j)*(i-k)*(i-l)*(j-k)*(j-l)*(k-l)/12$$

b. Implement the 4D permutation symbol `Eijkl` as function using the formula from a. Check, if e.g. $Eijk\ell(3,4,2,1) = -1$.

c. Define the 4D permutation symbol `eijkl` in MAXIMA using the 'exercise 15'-way, i.e. use a table of just enough prescribed values at quadruple index positions.

d. Define the 4D permutation symbol `eijkl` in MAXIMA using the build-in function `levi_civita()` from tensor package `itensor`.

7.2.4 a reconstruction of the `levi_civita` tensor

We now show a general procedure to calculate the `levi_civita` tensor using the determinant of permutated columns of the unit matrix, cf. $\triangleright\varepsilon_{i_1 i_2 ..}$. Here is the idea in $\mathbb{R}^3$:

$$\begin{bmatrix} 1&0&0\\0&1&0\\0&0&1 \end{bmatrix} \xrightarrow{231} \begin{bmatrix} 0&0&1\\1&0&0\\0&1&0 \end{bmatrix} \xrightarrow{det} 1 \longrightarrow \varepsilon_{231} = \texttt{levi\_civita}([2,3,1])$$

Starting with the unit matrix U, pick their columns 2, 3 and 1 in this sequence, build with these columns a new matrix, get their determinant - and that's the value of ε_{231}.

Here is this process in MAXIMA:

```
U : ident (3);
eijk(i,j,k) := determinant( matrix(U[i],U[j],U[k]) );
eijk(2,3,1);
```

YAMWI output:

1

Exercise 16. Redo exercise 7.2.3.b using the above procedure.

♡ The last exercise should convince you of the magnificent cool suitability of MAXIMA for doing tensor calculus without pain - and to present tensors visually to inspect their action.

7.3 Applications

As a proof of concept we show how the mathematical concepts of the determinant of a square matrix, the cross product of two vectors and the curl of a vector field can be defined and calculated using tensor technics with MAXIMA.

7.3.1 The Determinant

If $A = (a_{ij}) \in \mathbb{R}^{n\times n}$ is a n-by-n square matrix, the determinant is defined and calculated through the so-called LEIBNIZ formula using sigma notation[21],

$$\det(A) = \sum_{\sigma\in S_n} \operatorname{sgn}(\sigma) a_{1,\sigma_1} \cdots a_{n,\sigma_n}$$

Using '$\prod$' notation, this is in short $\det(A) = \sum_{\sigma\in S_n} \left(\operatorname{sgn}(\sigma) \prod_{i=1}^{n} a_{i,\sigma(i)}\right)$.
Written with the permutation symbol, the Leibniz formula is

$$\det(A) = \sum_{i_1,i_2,\ldots,i_n} \varepsilon_{i_1\cdots i_n} a_{1,i_1} \cdots a_{n,i_n} \tag{7.2}$$

where the sum is taken over all n-tuples of integers from $\{1,\ldots,n\}$.

This is a horrible formula, which no one really wants to compute this way by paper'n pencil. So let's do MAXIMA the work for us.

For the special case of a 3×3 matrix A we have for $det : \mathbb{R}^{3\times 3} \to \mathbb{R}^1$

$$\det(\mathbf{A}) = \sum_{i=1}^{3}\sum_{j=1}^{3}\sum_{k=1}^{3} \varepsilon_{ijk} a_{1i} a_{2j} a_{3k} \tag{7.3}$$

Here the permutation symbol ε_{ijk} acts as a filter, which sort out the proper ones of the whole 27 summands. Let's realize this in MAXIMA in three different ways.

1st definition of 3-by-3 determinant using LEIBNIZ formula and plain MAXIMA.
We use our self-made permutation symbol $Eijk$ and do the three sum's in (7.3) explicit.

```
A : matrix([1,2,3],[4,5,6],[7,8,8]);    /* a 3-by-3 matrix */
Eijk(i,j,k) := (j-i)*(k-i)*(k-j)/2;    /* permutation symbol as function */

Det(M) := sum(sum(sum(  Eijk(i,j,k)*M[1,i]*M[2,j]*M[3,k],      /* see (7.3) */
                                  i,1,3), j,1,3), k,1,3);
Det(A);
determinant(A);      /* check with build-in det-function */
```

MAXIMA output: 3
▷ Click here to run the script.

[21] see e.g. [12]

2nd definition of 3-by-3 determinant using LEIBNIZ formula and `levi_civita`.
We use the build-in permutation symbol and do the three sum's in (7.1) also explicit.

```
load(itensor)$          /* because we call levi_civita */
Det(M) := sum(sum(sum(  levi_civita([i,j,k])*M[1,i]*M[2,j]*M[3,k],
                                    i,1,3), j,1,3), k,1,3);

A : matrix([1,2,3],[4,5,6],[7,8,8]);
Det(A);
```

MAXIMA output: 3
▷ Click here to run the script.

3rd fully tensorial definition of 3-by-3 determinant;
here we have to be very precise w.r.t. using covariant and contravariant indices:

```
load(ctensor);
load(itensor);
imetric(g);
ct_coordsys( cartesian3d );
ishow( c = levi_civita([i,j,k],[])*A([1],[i])*A([2],[j])*A([3],[k]) )$
define( det(A), rhs( ic_convert(%) ));                          /* 1: */

for i thru dim do  for j thru dim do  for k thru dim do
for a thru dim do  for b thru dim do  for c thru dim do
           kdelta[i,j,k, a,b,c] : kdelta([i,j,k], [a,b,c]);     /* 2: */

det( matrix([1,2,3],[4,5,6],[7,8,8]) );
```

YAMWI output: 3

```
[%i5] ishow( levi_civita([i,j,k],[])*A([1],[i])*A([2],[j])*A([3],[k]) )$
[%t5]
                          i  j  k       1 2 3
                         A  A  A  kdelta
                          1  2  3       i j k
define( det(A), rhs( ic_convert(c=%)) );
[%o6]
             det(A) := sum(sum(sum(A     A     A
                                   1, i  2, j  3, k
              kdelta                , i, 1, dim), j, 1,
                    i, j, k, 1, 2, 3
             dim), k, 1, dim)
[%i7] /* 1: */

      for i thru dim do  for j thru dim do  for k thru dim do
      for a thru dim do for b thru dim do for c thru dim do
                kdelta[i,j,k, a,b,c] : kdelta([i,j,k], [a,b,c]);
[%o7]                                done
[%i8] /* 2: */

      det(matrix([1,2,3],[4,5,6],[7,8,8]));
[%o8]                                3
```

Comment. 1: assigns the RHS of the defining term $\sum_{i=1}^{3}\sum_{j=1}^{3}\sum_{k=1}^{3}\varepsilon_{ijk}A_1^iA_2^jA_3^k$ (remember the EINSTEIN summation convention! by writing A_i^j) to the term `det(A)`. Now 2: assigns numerical values to the `kdelta` list-indexed expression that `ic_convert()` blindly generates.[22] The construction is analog to that in §6.2.4th.

▷ Click here to run this script.

Exercise 17. Define a MAXIMA function `Det2` analog to the example above and calculate the determinant of matrix $A = \left[\begin{smallmatrix}1&2\\7&8\end{smallmatrix}\right]$.

7.3.2 Cross product

Given two linearly independent vectors a and b in $\mathbb{R}^3$, the cross product $a \times b$ is a vector, that is perpendicular to both a and b, i.e. therefore perpendicular to the plane containing them. Using column vector notation, we have the explicit defining formula for $c = a \times b$:

$$\begin{bmatrix}c_1\\c_2\\c_3\end{bmatrix} = \begin{bmatrix}a_2b_3 - a_3b_2\\a_3b_1 - a_1b_3\\a_1b_2 - a_2b_1\end{bmatrix}$$

In any basis, the cross-product $a \times b$ is given by the tensorial formula

$$(a \times b)_k = \varepsilon_{ijk}a^ib^j$$

EINSTEIN ... $\downarrow$ ... expanded

$$(a \times b)_k = \sum_{i=1}^{3}\sum_{j=1}^{3}\varepsilon_{ijk}a^ib^j \quad (7.3.2)$$

i.e. with the suppressed summation expanded, we get the corresponding formula for the translation of the cross product term into MAXIMA `itensor` code. Therein ε_{ijk} is the permutation symbol `levi_civita([i,j,k])`, which respects the positions of the indices:

1st implementation of cross product formula with `levi_civita`.

```
U : ident (3);
load(itensor)$              /* because we call levi_civita */
cross(A,B) := sum(sum(sum( levi_civita([i,j,k])*U[i]*A[j]*B[k],
                                          i,1,3), j,1,3), k,1,3);
A : [4,5,6];
B : [1,2,3];
cross(A,B);
```

[22] Note, that `levi_civita(()` is constructed using kdelta δ_i^j terms. Therefore the need to evaluate the kdelta terms. We will see this art of evaluation of terms often, when going from `itensor` to `ctensor` package.

MAXIMA output: `[3, -6, 3]`
▷ Click here to run this script.

Exercise 18. Use our self-made permutation symbol $Eijk$ and redo 1^{st} to bypass the call to `itensor`.

Exercise 19. If you only need the first component `axb1` of the cross product of a and b, you can do: `axb1 := sum( sum( Eijk( 1 ,j,k)*a[j]*b[k] , i...)`.
Verify it. Use `Eijk(i,j,k)` from §7.3.1.1.

2^{st} implementation of cross product formula with `levi_civita`.
Each individual component of (7.2) is programmed as a single function $\text{cross}_k(M)$:

```
load(itensor)$
cross1(M) := sum(sum( levi_civita([i,j, 1 ])*M[1,i]*M[2,j], i,1,3), j,1,3);
cross2(M) := sum(sum( levi_civita([i,j, 2 ])*M[1,i]*M[2,j], i,1,3), j,1,3);
cross3(M) := sum(sum( levi_civita([i,j, 3 ])*M[1,i]*M[2,j], i,1,3), j,1,3);

cross(M) := [cross1(M), cross2(M),  cross3(M)] ;

A : matrix([1,2,3],[4,5,6]);
cross(A);
```

MAXIMA output: `[3, -6, 3]`
▷ Click here to run this script.

Exercise 20. (* An Preview of `contract`)
The individual components $\text{cross}_k(A, B)$ of the 2^{nd} version of cross may be programmed fully tensorial using function `contract` of package `itensor` to do the silent summations. Here is $\text{cross}_1(A, B)$:

```
load("itensor")$
ishow( c1 = levi_civita([],[i,j,1]) * A([i],[]) * B([j],[]))$
ishow( contract( expand( ev(%, kdelta))))$
define( cross1(A,B), rhs( ic_convert(%)) );

A(u,v):=A[u[1]]$
B(u,v):=B[u[1]]$
cross1( [1,2,3], [4,5,6]);
```

▷ Click here to run this script. [23]
Then write cross_2 and cross_3 and now do

```
cross(x,y) := [cross1( [x[1],x[2],x[3]], [y[1],y[2],y[3]]),
               cross2( [x[1],x[2],x[3]], [y[1],y[2],y[3]]),
               cross3( [x[1],x[2],x[3]], [y[1],y[2],y[3]])];
cross([1,2,3], [4,5,6]);
```

[23] In case you want more and detailed explanation ▷1.

YAMWI output:

```
[%i4]  ishow( c([],[k]) = 'levi_civita([],[i,j,k]) * a([i],[]) * b([j],[]))$
[%t4]                          k              i j k
                             c  = levi_civita      a  b
                                                    i  j
[%i5]  ishow( ev(%, levi_civita));
[%t5]                            k          i j k
                               c  = kdelta       a  b
                                            1 2 3  i  j
[%o5]                 c([], [k]) = kdelta([1, 2, 3], [i, j, k])
                      a([i], []) b([j], [])
[%i6]  ishow( expand( ev(%, kdelta)))$
[%t6]                        k          i        j        k
                           c  = kdelta  kdelta  kdelta  a  b
                                  1        2        3  i  j
                                j        i        k
                           - kdelta  kdelta  kdelta  a  b
                                                  ...
       ishow( contract(%))$
[%t7]                        k                k                k
                           c  = a  b  kdelta  - b  a  kdelta
                                1  2        3    1  2        3
                                   k             k
                           - a  kdelta  b  + kdelta  a  b
                              1         2  3         1  2  3
                                   k             k
                           + b  kdelta  a  - kdelta  b  a
                              1         2  3         1  2  3
```

Comment. (%t5) shows how the *levi_civita* symbol of line (%t4) is expressed via the general *kdelta* tensor, which is then evaluated and expanded in many usual δ_i^j products. Line (%t7) shows the contraction (summation), where the first component $cross_1$ of the cross product may be read of setting $k = 1$.

∘ The definition of function *cross* in 2^{nd} and 1^{st} views it as *vector-valued* function $cross : \mathbb{R}^3 \times \mathbb{R}^3 \to \mathbb{R}^3$, whereas *det* is *scalar* valued.

Looking back, we have programmed the mathematical *cross* product in steps of increasing abstractness in MAXIMA. Starting with the explicit summations to code whole and components of *cross* until a first view to *contract* to do the summations w.r.t. the EINSTEIN summation convention.

7.3.3 Curl

curl F [24] is a vector operator that is used in vector calculus to describe the circulation of a 3D differentiable vector field $F\colon \mathbb{R}^3 \to \mathbb{R}^3$, given through its three component functions $F(x,y,z) = (F_x(x,y,z), F_y(x,y,z), F_z(x,y,z))$.
We cite the definition:

$$(curl\, F)(x,y,z) = \left(\frac{\partial F_z}{\partial y} - \frac{\partial F_y}{\partial z}, \frac{\partial F_x}{\partial z} - \frac{\partial F_z}{\partial x}, \frac{\partial F_y}{\partial x} - \frac{\partial F_x}{\partial y}\right) \tag{7.4}$$

The equation for each component $(curl\, F)_k$ is obtained by exchanging each occurrence of a subscript y, z, x in cyclic permutation $(y,z) \to (z,x) \to (x,y)$ by focussing at the denominators. – We implement *curl* in MAXIMA in two ways.

1st classic vectorial definition[25] of *curl* with plain MAXIMA

We implement the classic definition from above straight ahead in MAXIMA and test it with the vector field $F(x,y,z) = (xy, -\sin(z), 1)$:

```
curl(F,x,y,z):=[ diff(F[3],y)-diff(F[2],z),
                 diff(F[1],z)-diff(F[3],x),
                 diff(F[2],x)-diff(F[1],y) ]$

F(x,y,z):=[x*y,-sin(z),1];
curl(F(x,y,z),x,y,z);
```

MAXIMA output: `[cos(z), 0, -x]`
▷ Click here to run the code.

2nd tensorial version of *curl* using `c/i/tensor`

If we write $F = (F_1, F_2, F_3)$ as a function of position $x = (x^1, x^2, x^3)$ in $\mathbb{R}^3$ using index notion, then the ith component of the curl of F equals (summation suppressed according the EINSTEIN summation convention!)

$$(curl\, \mathbf{F})_i(\mathbf{x}) = \varepsilon^{ijk} \frac{\partial}{\partial x^j} F_k(\mathbf{x}) \tag{7.5}$$

in tensorial notation, which follows from the cross product definition in 7.3.2, substituting components of the gradient vector.
Index i is 'free' in the defining term (7.3) in the sense that it is not seen twice on the RHS of the tensor term. But index j and k are to be seen twofold on the RHS - each one once as superscript ('upper index') and once as subscript ('lower index'). Therefore

[24] In classic scientific literature in Europe, the alternative notation $rotF$ (read: rotation F) is used.
[25] cf. Barth

there are 2 implicit hidden summation processes involved in accordance with the EINSTEIN summation convention. So we have two phases $A \to B$ to construct the term in MAXIMA:

$$\varepsilon_{ijk} \cdot \frac{\partial}{\partial x^j} F^k(\mathbf{x})$$
$$A \downarrow \textit{ sum over } k, \quad \textit{i.e. 'contract'}$$
$$B \downarrow \textit{ sum over } j \quad \textit{i.e. 'contract'}$$

We then have

```
/*-- contributed by V. Toth --*/
load(itensor);
load(ctensor);
dim : 3;
curlF : cF([],[i]) = levi_civita([],[i,j,k]) * idiff(F([k],[]), j); /*1:*/
curlF : ic_convert(curlF);                                          /*2:*/
cF : [0,0,0];                                                       /*3:*/

for i thru dim do   for j thru dim do  for k thru dim do
for a thru dim do   for b thru dim do  for c thru dim do
      kdelta[i,j,k, a,b,c] : kdelta([i,j,k],[a,b,c]);          /*4:*/

ev( curlF,                                                   /*5:*/
    F : [x*y,-sin(z),1],
    ct_coords : [x,y,z],
    kdelta);
cF;
```

Comment. Line 1: writes the defining tensorial term $(curl\,\mathbf{F})_i(\mathbf{x}) = \varepsilon^{ijk}\frac{\partial}{\partial x^j}F_k(\mathbf{x})$ as itensor term using the

LEXICON	*Math*	itensor
	$(curl\,\mathbf{F})_i$	`cF([],[i])`
	ε^{ijk}	`levi_civita([],[i,j,k])`
	$\frac{\partial}{\partial x^j}F_k$	`idiff(F([k],[])`

and the differential operator `idiff` of `itensor` package. Be warned: you must use the equal sign '=' here, which is reserved for equations, i.e. do _not_ use the assignment operator ':' here. In 2: we convert the itensor equation 1: to an evaluable ctensor expression. In 3: we define cF as an array – otherwise it'd default to a list. 4: is tricky: we define the generalized *kdelta* itensor object for the use in ctensor, because ctensor does not know about *kdelta*! 5: evaluates all objects for the concrete example.

MAXIMA output: `[cos(z), 0, -x]`

▷ Click here to RUN the code.

Exercise 21. Write a function `define(cross(F), ..)` along the lines of §7.3.2 3^{rd}.

Exercise 22. Calculate the the curl of the following vector fields [Mardsen, p. 914 ff].
a. $F(x, y, z) = (e^z, -\cos(xy), z^3 y)$.
b. $F(x, y, z) = (xy\cos(x), -yz\sin(x), -xy\tan(y))$.
c. $F(x, y, z) = (yz, -xz, xy) \cdot 1/(x^2 + y^2 + z^2)$.
d. $F(x, y, z) = (ye^z, xe^z, xye^z)$.

Exercise 23. Let $f\colon \mathbb{R}^3 \to \mathbb{R}$ be defined through $f(x, y, z) = x^2yz^3$ and $A\colon \mathbb{R}^3 \to \mathbb{R}^3$ be defined through $A(x, y, z) = (xz, -y^2, 2x^2y)$.
Calculate curl(A) and curl$(f \cdot A)$, cf. [30, p. 151].

Exercise 24. Let f be $f(x, y, z) = xy + yz + zx$ and $A(x, y, z) = (x^2y, y^2z, z2^{)}$.
Calculate curl(A) and curl$(f \cdot A)$ at the point $P = (3, -1, 2)$, cf. [30, p. 151].

Exercise 25. If $A\colon \mathbb{R}^3 \to \mathbb{R}^3$ is defined through $A(x, y, z) = (3xz^2, -yz, x + 2z)$, calculate curl curl$(A)$, cf. [30, p. 158].

8 KRONECKER product

8.1 `outer`

At first we define the `outer` product, which implements the tensor operator $\overset{outer}{\otimes}$ of tensor algebra. We define the outer product only in the special case of vectors.

8.1.1 Definition : `outer`

We quote WIKIPEDIA's outer definition.

Given two vectors $\mathbf{u} = \begin{bmatrix} u_1 \\ u_2 \\ \vdots \\ u_m \end{bmatrix}$ and $\mathbf{v} = \begin{bmatrix} v_1 \\ v_2 \\ \vdots \\ v_n \end{bmatrix}$ of size $m \times 1$ and $n \times 1$, their *outer product*, denoted $\mathbf{u} \overset{outer}{\otimes} \mathbf{v}$ is defined as the $m \times n$ matrix obtained by multiplying each element of $\mathbf{u}$ by each element of $\mathbf{v}$:

$$\mathbf{u} \overset{outer}{\otimes} \mathbf{v} := \begin{bmatrix} u_1v_1 & u_1v_2 & \dots & u_1v_n \\ u_2v_1 & u_2v_2 & \dots & u_2v_n \\ \vdots & \vdots & \ddots & \vdots \\ u_mv_1 & u_mv_2 & \dots & u_mv_n \end{bmatrix}$$

In tensorial index notation: $(\mathbf{u} \overset{outer}{\otimes} \mathbf{v})_{ij} = u_i v_j$

- The outer product $\mathbf{u} \otimes \mathbf{v}$ is equivalent to a matrix multiplication $\mathbf{u}\mathbf{v}^{\mathrm{T}}$, provided that $\mathbf{u}$ is represented as a $m \times 1$ column vector and $\mathbf{v}$ as a $n \times 1$ column vector (which makes $\mathbf{v}^{\mathrm{T}}$ a row vector).

The concept of the outer product $\overset{outer}{\otimes}$ is best understood by looking at several examples.

8.1.2 Examples of the use of `outer`

Example 10. (the outer product of two vectors)

```
u : matrix([1], [2], [3]);                  /* a column vector       */
v : transpose(matrix([4,5]));              /* another column vector */
                    /* Compute outer product  by (u * transpose(v)) */
outer_prod: u . transpose(v);

infix ("##", 99, 99);                   /* define infix operator  */
"##"(u, v) := u . transpose(v);       /* named '##' for easier input */

u##v;                                /* use the infix operator */
```

▷ Click here to run the code.

xMAXIMA output:

```
[ 4   5  ]
[ 8   10 ]
[ 12  15 ]
```

Excursus: Let's make a short detour and use for the moment the CAS `EigenMath`, cf. [44], which has implemented a function `outer` working also for matrices and multidimensional arrays as inputs. This function also respects the block structure of their inputs, which maybe support a first understanding. As we will see in §8.2, MAXIMA's `kronecker_product` is equivalent to $\texttt{outer}|_{Eigenmath}$, but suppresses the block structure and flattens it's output.

Example 11. (the outer product of two matrices with *arbitrary* dimensions)

```
# EIGENMATH
A = ((1, 2, 3),( 4, 5, 6))  -- a 2-by-3 matrix
A
B = ((1,1),(1,1))           -- ones(2), a 2-by-2 matrix
B
outer(A,B)                  -- a (2,3,2,2) 'tensor' (multidim. matrix)
```

▷ Click here to run the example

`EigenMath` output:

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}$$

$$B = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}$$

$$\begin{bmatrix} \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix} & \begin{bmatrix} 2 & 2 \\ 2 & 2 \end{bmatrix} & \begin{bmatrix} 3 & 3 \\ 3 & 3 \end{bmatrix} \\ \begin{bmatrix} 4 & 4 \\ 4 & 4 \end{bmatrix} & \begin{bmatrix} 5 & 5 \\ 5 & 5 \end{bmatrix} & \begin{bmatrix} 6 & 6 \\ 6 & 6 \end{bmatrix} \end{bmatrix}$$

Obviously $\texttt{outer}|_{Eigenmath}$ respects the block structure of both input matrices.

Example 12. (the outer product of two tensors with *arbitrary* dimensions)

```
# EIGENMATH
i = quote(i)          -- clear name i (otherwise i^2=1 ;)

T = zero(2,2,2)       -- a (2,2,2) tensor
for(i,1,2,  for(j,1,2, for(k,1,2,  T[i,j,k] = i^2*j*k))) -- initialize T

S = zero(2,2,2,2)     -- a (2,2,2,2) tensor
for(i,1,2, for(j,1,2, for(k,1,2, for(l,1,2, S[i,j,k,l] = i^2*j*k*l))))

T
```

```
S
TT = outer(T,T)
TT
TS=outer(T,S)
TS
```

▷ Click here to run this example.
`EigenMath` output:

$$T = \begin{bmatrix} \begin{bmatrix} 1 & 2 \\ 2 & 4 \end{bmatrix} \\ \begin{bmatrix} 4 & 8 \\ 8 & 16 \end{bmatrix} \end{bmatrix}$$

$$S = \begin{bmatrix} \begin{bmatrix} 1 & 2 \\ 2 & 4 \end{bmatrix} & \begin{bmatrix} 2 & 4 \\ 4 & 8 \end{bmatrix} \\ \begin{bmatrix} 4 & 8 \\ 8 & 16 \end{bmatrix} & \begin{bmatrix} 8 & 16 \\ 16 & 32 \end{bmatrix} \end{bmatrix}$$

$$T_T = \begin{bmatrix} \begin{bmatrix} \begin{bmatrix} 1 & 2 \\ 2 & 4 \end{bmatrix} & \begin{bmatrix} 4 & 8 \\ 8 & 16 \end{bmatrix} \\ \begin{bmatrix} 2 & 4 \\ 4 & 8 \end{bmatrix} & \begin{bmatrix} 8 & 16 \\ 16 & 32 \end{bmatrix} \end{bmatrix} & \begin{bmatrix} \begin{bmatrix} 2 & 4 \\ 4 & 8 \end{bmatrix} & \begin{bmatrix} 8 & 16 \\ 16 & 32 \end{bmatrix} \\ \begin{bmatrix} 4 & 8 \\ 8 & 16 \end{bmatrix} & \begin{bmatrix} 16 & 32 \\ 32 & 64 \end{bmatrix} \end{bmatrix} \\ \begin{bmatrix} \begin{bmatrix} 4 & 8 \\ 8 & 16 \end{bmatrix} & \begin{bmatrix} 16 & 32 \\ 32 & 64 \end{bmatrix} \\ \begin{bmatrix} 8 & 16 \\ 16 & 32 \end{bmatrix} & \begin{bmatrix} 32 & 64 \\ 64 & 128 \end{bmatrix} \end{bmatrix} & \begin{bmatrix} \begin{bmatrix} 8 & 16 \\ 16 & 32 \end{bmatrix} & \begin{bmatrix} 32 & 64 \\ 64 & 128 \end{bmatrix} \\ \begin{bmatrix} 16 & 32 \\ 32 & 64 \end{bmatrix} & \begin{bmatrix} 64 & 128 \\ 128 & 256 \end{bmatrix} \end{bmatrix} \end{bmatrix}$$

Here is tensor TS:

$$T_S = \begin{bmatrix} \begin{bmatrix} \begin{bmatrix} \begin{bmatrix} 1 & 2 \\ 2 & 4 \end{bmatrix} & \begin{bmatrix} 2 & 4 \\ 4 & 8 \end{bmatrix} \\ \begin{bmatrix} 4 & 8 \\ 8 & 16 \end{bmatrix} & \begin{bmatrix} 8 & 16 \\ 16 & 32 \end{bmatrix} \end{bmatrix} & \begin{bmatrix} \begin{bmatrix} 2 & 4 \\ 4 & 8 \end{bmatrix} & \begin{bmatrix} 4 & 8 \\ 8 & 16 \end{bmatrix} \\ \begin{bmatrix} 8 & 16 \\ 16 & 32 \end{bmatrix} & \begin{bmatrix} 16 & 32 \\ 32 & 64 \end{bmatrix} \end{bmatrix} \\ \begin{bmatrix} \begin{bmatrix} 2 & 4 \\ 4 & 8 \end{bmatrix} & \begin{bmatrix} 4 & 8 \\ 8 & 16 \end{bmatrix} \\ \begin{bmatrix} 8 & 16 \\ 16 & 32 \end{bmatrix} & \begin{bmatrix} 16 & 32 \\ 32 & 64 \end{bmatrix} \end{bmatrix} & \begin{bmatrix} \begin{bmatrix} 4 & 8 \\ 8 & 16 \end{bmatrix} & \begin{bmatrix} 8 & 16 \\ 16 & 32 \end{bmatrix} \\ \begin{bmatrix} 16 & 32 \\ 32 & 64 \end{bmatrix} & \begin{bmatrix} 32 & 64 \\ 64 & 128 \end{bmatrix} \end{bmatrix} \end{bmatrix} \\ \begin{bmatrix} \begin{bmatrix} \begin{bmatrix} 4 & 8 \\ 8 & 16 \end{bmatrix} & \begin{bmatrix} 8 & 16 \\ 16 & 32 \end{bmatrix} \\ \begin{bmatrix} 16 & 32 \\ 32 & 64 \end{bmatrix} & \begin{bmatrix} 32 & 64 \\ 64 & 128 \end{bmatrix} \end{bmatrix} & \begin{bmatrix} \begin{bmatrix} 8 & 16 \\ 16 & 32 \end{bmatrix} & \begin{bmatrix} 16 & 32 \\ 32 & 64 \end{bmatrix} \\ \begin{bmatrix} 32 & 64 \\ 64 & 128 \end{bmatrix} & \begin{bmatrix} 64 & 128 \\ 128 & 256 \end{bmatrix} \end{bmatrix} \\ \begin{bmatrix} \begin{bmatrix} 8 & 16 \\ 16 & 32 \end{bmatrix} & \begin{bmatrix} 16 & 32 \\ 32 & 64 \end{bmatrix} \\ \begin{bmatrix} 32 & 64 \\ 64 & 128 \end{bmatrix} & \begin{bmatrix} 64 & 128 \\ 128 & 256 \end{bmatrix} \end{bmatrix} & \begin{bmatrix} \begin{bmatrix} 16 & 32 \\ 32 & 64 \end{bmatrix} & \begin{bmatrix} 32 & 64 \\ 64 & 128 \end{bmatrix} \\ \begin{bmatrix} 64 & 128 \\ 128 & 256 \end{bmatrix} & \begin{bmatrix} 128 & 256 \\ 256 & 512 \end{bmatrix} \end{bmatrix} \end{bmatrix} \end{bmatrix}$$

Exercise 26. Let $A = ((x, y), (z, u))$ and $B = ((a, b), (c, d))$.
Calculate $A \overset{outer}{\otimes} B$, $B \overset{outer}{\otimes} (A \overset{outer}{\otimes} B)$ and $(A \overset{outer}{\otimes} B) \overset{outer}{\otimes} (A \overset{outer}{\otimes} B)$ using the $\texttt{outer}|_{Eigenmath}$. Imagine by mind what output is to be expected.

8.2 `Kronecker` product

MAXIMA's `kronecker_product` function implements the tensor operator $\otimes$ of tensor algebra, i.e. `kronecker_product(A,B)` returns the Kronecker tensor product of tensors (e.g. matrices) A and B. The Kronecker product is best understood by thinking about block matrices. If A is a $n \times p$ matrix and B is a $m \times q$ matrix, then the Kronecker product $A \otimes B$ is a block matrix that is build from blocks of B, where each block is multiplied by an element of A, cf. [45].

8.2.1 Definition of `kronecker product`

Let A be an $m \times n$ matrix and let B an $p \times q$ matrix.
Then the *Kronecker product of A and B*, symbolically written $A \otimes B$, is the $m \cdot p \times n \cdot q$ matrix explicitly defined by

$$A \otimes B := (a_{ij} \cdot B) = \begin{pmatrix} a_{11}B & \cdots & a_{1n}B \\ \vdots & \ddots & \vdots \\ a_{m1}B & \cdots & a_{mn}B \end{pmatrix} \tag{8.1}$$

$$= \texttt{kronecker\_product}(\texttt{A}, \texttt{B}) \quad \text{in Maxima} \tag{8.2}$$

In words: If A is an m-by-n matrix and B is a p-by-q matrix, then `kronecker_product(A,B)` is an $m \cdot p$-by-$n \cdot q$ matrix, which is calculated by taking all possible products between the elements of A and those of B, i.e. the Kronecker tensor product of A and B is a large matrix formed by multiplying B by each element of A.

8.2.2 Examples of the use of `kronecker_product`

Example 13.

$$\begin{pmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{pmatrix} \otimes \begin{pmatrix} 7 & 8 \\ 9 & 0 \end{pmatrix} = \begin{pmatrix} 1 \cdot \begin{pmatrix} 7 & 8 \\ 9 & 0 \end{pmatrix} & 2 \cdot \begin{pmatrix} 7 & 8 \\ 9 & 0 \end{pmatrix} \\ 3 \cdot \begin{pmatrix} 7 & 8 \\ 9 & 0 \end{pmatrix} & 4 \cdot \begin{pmatrix} 7 & 8 \\ 9 & 0 \end{pmatrix} \\ 5 \cdot \begin{pmatrix} 7 & 8 \\ 9 & 0 \end{pmatrix} & 6 \cdot \begin{pmatrix} 7 & 8 \\ 9 & 0 \end{pmatrix} \end{pmatrix} = \begin{pmatrix} 7 & 8 & 14 & 16 \\ 9 & 0 & 18 & 0 \\ 21 & 24 & 28 & 32 \\ 27 & 0 & 36 & 0 \\ 35 & 40 & 42 & 48 \\ 45 & 0 & 54 & 0 \end{pmatrix}$$

Calculated in MAXIMA:

```
kronecker_product( matrix([1,2],[3,4],[5,6]), matrix([7,8],[9,0]) );
```

MAXIMA output:

cf. hand-calculated example 13.

We now repeat the examples of `outer` to contrast them with the results of `kronecker`.

Example 14. (the Kronecker product of two vectors)

```
/* a: */
u : matrix([1,2,3]);
v : matrix([3,2,1]);
kronecker_product(u,v);

/* b: */
u : transpose(matrix([1,2,3]));
v : transpose(matrix([3,2,1]));
kronecker_product(u,v);

/* c: */
u : transpose(matrix([1,2,3]));
v : matrix([3,2,1]);
kronecker_product(u,v);

/* d: */
u : matrix([1,2,3]);
v : transpose(matrix([3,2,1]));
kronecker_product(u,v);
```

▷ Click here to run the code..
xMaxima output:

```
a:   [ 3  2  1  6  4  2  9  6  3 ]

b:   [ 3 ]
     [ 2 ]
     [ 1 ]
      ...
     [ 6 ]
     [ 3 ]

c:   [ 3  2  1 ]
     [ 6  4  2 ]
     [ 9  6  3 ]

d:   [ 3  6  9 ]
     [ 2  4  6 ]
     [ 1  2  3 ]
```

In contrast to `outer` the output of `kronecker_product` has stacked the rows to one long column.

Example 15. (the Kronecker product of two matrices with *arbitrary* dimensions)

```
infix ("##", 99, 99);
"##"(a, b) := kronecker_product(a,b); /* Kronecker\_product abbreviated */
                                      /* as infix operator named '##'   */

A : matrix([1, 2, 3],[ 4, 5, 6]);
B : matrix([1,1],[1,1]);
A ## B;
```

▷ Click here to run the code.

MAXIMA output, cf. example 11:

$$\begin{bmatrix} 1 & 1 & 2 & 2 & 3 & 3 \\ 1 & 1 & 2 & 2 & 3 & 3 \\ 4 & 4 & 5 & 5 & 6 & 6 \\ 4 & 4 & 5 & 5 & 6 & 6 \end{bmatrix}$$

In contrast to `outer` the output of `kronecker_product` has cleared the brackets of the inner matrices to give one single matrix. Obviously `kronecker_product` 'flattens' the block structure of both input matrices.

Exercise 27. Let $A = \left[\begin{smallmatrix} x & y \\ z & u \end{smallmatrix}\right]$ and $B = \left[\begin{smallmatrix} a & b \\ c & d \end{smallmatrix}\right]$.
Calculate $A \otimes B$, $B \otimes (A \otimes B)$ and $(A \otimes B) \otimes (A \otimes B)$

Exercise 28. Write a function `ones(n)`, which returns a n-by-n matrix whose all entries are 1, e.g. $ones(2)$ is $\left[\begin{smallmatrix} 1 & 1 \\ 1 & 1 \end{smallmatrix}\right]$.

8.3 Applications of `kronecker` $\otimes$

8.3.1 The direct sum $\oplus$

Let A be an $m \times n$ matrix and B an $p \times q$ matrix. Then the *direct sum of A and B*, symbolically written $A \oplus B$, is the $m \cdot p \times n \cdot q$ matrix explicitly defined by

$$A \oplus B := \begin{pmatrix} A & O \\ O & B \end{pmatrix} \tag{8.3}$$

i.e. for any arbitrary matrices A and B the direct sum (aka tensor sum) is defined as the block diagonal matrix of A and B. Both zero matrices O are of appropriate dimension.

Example 16. (cf. STEEB [32, p. 7], [31, p. 28])

$$\begin{pmatrix} 1 & 2 & 3 \end{pmatrix} \oplus \begin{pmatrix} 4 & 5 \\ 6 & 7 \end{pmatrix} = \begin{pmatrix} \begin{pmatrix} 1 & 2 & 3 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix} & \begin{pmatrix} 0 & 0 \\ 4 & 5 \\ 6 & 7 \end{pmatrix} \end{pmatrix} = \begin{pmatrix} 1 & 2 & 3 & 0 & 0 \\ 0 & 0 & 0 & 4 & 5 \\ 0 & 0 & 0 & 6 & 7 \end{pmatrix}$$

We show how the direct sum could be defined in MAXIMA using function `kronecker_`$\otimes$.

```
A : matrix([1,2],[3,4]);
B : matrix([4,5],[6,7]);

U : matrix([1,0],[0,0]);  /*-- Upper */
L : matrix([0,0],[0,1]);  /*-- Lower */

kronecker_product(U,A);

directSum(A,B) :=  kronecker_product(U,A) + kronecker_product(L,B);
directSum(A,B);
```

▷ Click here to run this script.
MAXIMA output:

$$\begin{bmatrix} 1 & 2 & 0 & 0 \\ 3 & 4 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

$$\begin{bmatrix} 1 & 2 & 0 & 0 \\ 3 & 4 & 0 & 0 \\ 0 & 0 & 4 & 5 \\ 0 & 0 & 6 & 7 \end{bmatrix}$$

The first matrix is the result of the call `kronecker_product(U,A)` and the second matrix is the result of the call `directSum(A,B)`

Exercise 29. Calculate the direct sum of A and B from examples 13/15 with `directSum(A,B)`.

8.3.2 Construction of the basis of $\mathbb{R}^4$ from the basis of $\mathbb{R}^2$, cf. [32, p. 7]

Let $u = \begin{bmatrix}1\\0\end{bmatrix}, v = \begin{bmatrix}0\\1\end{bmatrix}$ be the canonical standard basis in $\mathbb{R}^2$.
Verify, that $(u \otimes u, u \otimes v, v \otimes u, u \otimes u)$ is the standard basis in $\mathbb{R}^4$.

We use MAXIMA's `kronecker_product`.

```
kron(A,B) := kronecker_product(A,B);

u : matrix([1],[0]);
v : matrix([0],[1]);

/* -- Basis R^4 */
kron(u,u);
kron(u,v);
kron(v,u);
kron(v,v);
```

▷ Click here to run this script.
MAXIMA output: $[1,0,0,0]^t, ..., [0,0,0,1]^t$

Exercise 30. Construct an orthonormal basis of $\mathbb{C}^4$ with the help of `kronecker_product`. Hint: start with $u = 1/\sqrt{2} \cdot \left[\begin{smallmatrix}1\\ i\end{smallmatrix}\right]$ and $u = 1/\sqrt{2} \cdot \left[\begin{smallmatrix}1\\ -i\end{smallmatrix}\right]$.

Exercise 31. (cf. [32, p. 112]) A basis in the Hilbert space $M_2(\mathbb{C})$ of 2-by-2 matrices is given by the PAULI spin matrices

$$\sigma_0 = \begin{bmatrix}1 & 0\\ 0 & 1\end{bmatrix}, \sigma_1 = \begin{bmatrix}0 & 1\\ 1 & 0\end{bmatrix}, \sigma_2 = \begin{bmatrix}0 & -i\\ i & 0\end{bmatrix}, \sigma_3 = \begin{bmatrix}1 & 0\\ 0 & -1\end{bmatrix}$$

including the 2-by-2 identity matrix. Use this orthonormal basis and the Kronecker products $\sigma_j \otimes \sigma_k$ with $j, k = 0, 1, 2, 3$ to find a basis in the Hilbert space $M_4(\mathbb{C})$.

Solution:

```
kron(A,B) := kronecker_product(A,B);
/*-- Pauli spins is basis in M2(C) */
s[0] : matrix([1,0],[0,1]);
s[1] : matrix([0,1],[1,0]);
s[2] : matrix([0,-%i],[%i,0]);
s[3] : matrix([1,0],[0,-1]);
for j:0 thru 3 do
    for k:0 thru 3 do
        print( kron(s[j],s[k]) ); /*-- Basis M4(C) */
```

▷ Click here to run this script.

9 Contraction

The *contraction* of a tensor acts by setting equal a pair of indices[26] of the tensor to each other and summed over. In EINSTEIN summation convention this summation is implicit in the notation. The result is a new tensor with order reduced by 2, cf. [49].
Contraction of tensors is heavily used in Differential Geometry or in General Relativity. Tensor contraction is a generalization of the trace of a matrix.

9.1 Definition of `contract`

Let $\mathbf{T} = T^i{}_j$ be a tensor. Then its *contraction* is defined by

$$T^i{}_j\delta_i{}^j := T^j{}_j = \sum_{j=1}^{n} T^j{}_j = T^1{}_1 + \cdots + T^n{}_n \tag{9.1}$$

In MAXIMA's `itensor` package this is expressed with the call of `contract()`.

- A general contraction is denoted by labeling one (covariant, lower) index and one (contravariant, upper) index with the same letter, summation over that index being implied by the summation convention, i.e. the call of `contract`. The resulting contracted tensor inherits the remaining indices of the original tensor, cf. [49].

- For example, contracting a tensor T of type (2,2) on the second and third indices (i.e. b) to create a new tensor U of type (1,1) is written as

$$T^{ab}{}_{bc} = \sum_b T^{ab}{}_{bc} = T^{a1}{}_{1c} + T^{a2}{}_{2c} + \cdots + T^{an}{}_{nc} = U^a{}_c$$

Example 17. Basic Contraction Process with `itensor` package

CONTRACTION	`itensor`
1 Load the package.	`load("itensor");`
2 inform itensor that g is the metric tensor.	`imetric(g);`
3 enable index raising/lowering via metric.	`defcon(g);`
4 let contract performs the EINSTEIN-summation.	`contract(expression);`

Let's contract a tensor (vector) using the metric:

```
load("itensor")$
imetric(g)$                          /* set metric name to g */
defcon(g)$                          /* define contraction for g */
ishow(g([i,j],[])*v([],[j]))$   /* display g_ij * v^j */
contract(%);                       /* contract index j : g_ij * v^j -> v_i */
```

[26] one a subscript (upper) index, the other a superscript (lower) index

▷ Click here to run this script.

YAMWI output:

```
[%i4] ishow(g([i,j],[])*v([],[j]));
[%t4]
                    j
                 v  g
                      i j
[%o4]   v([], [j]) g([i, j], [])
[%i5] contract(%);
[%o5]          v([i], [])
[%i6] ishow(%);
[%t6]
                    v
                     i
```

Exercise 32. a. Contact tensor T^{ab} along index b using metric g.
b. Contact tensor T^{ab}_{bc} along index b using metric g.
Hint: `ishow(g([b,c],[])*T([],[a,b]))`

9.2 Examples

Example 18. (orbital velocity in circular motion - a second view at *cross*)
We quote from the German help for `levi_civita` function. For circular motion, the orbital velocity $\vec{w}$ is the cross product of the angular velocity $\vec{v}$ and the position vector $\vec{r}$. Thus, we have $\vec{w} = \vec{v} \times \vec{r}$:

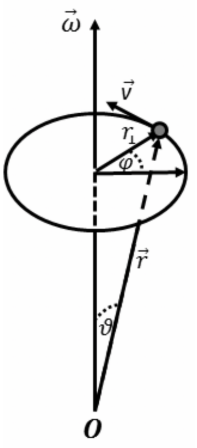


Figure 18: ⊙: circular motion of point • with orbital velocity $\vec{w}$, angular velocity $\vec{v}$ and position vector $\vec{r}$.

In any basis, the a-th component of the orbital velocity $\vec{w}$ is given by the tensorial formula, cf. (7.3.2):

$$w^a = \varepsilon^{abc} v_b r_c \tag{9.2}$$

Task: calculate `cross([1,2,3],[4,5,6])` using tensor technics.

1st First, a tensorial notation for the first component of cross product using the levi-civita symbol is introduced. The expression is then simplified for the first component to the familiar definition of the cross product. Here is a variant of exercise 20 using `contract`:

```
load(itensor)$
ishow( w([a],[]) = 'levi_civita([a,b,c],[])*v([],[b])*r([],[c]))$   /*1:*/
ishow( subst([a=1],%))$                                             /*2:*/
ishow( ev(%, levi_civita))$                                         /*3:*/
ishow( expand( ev(%, kdelta)))$                                     /*4:*/
ishow( contract(%))$                                                /*5:*/
define( cross1(v,r), rhs( ic_convert(c=%)) );                       /*6:*/

cross1(v,r) := v[2]*r[3]-r[2]*v[3];                                 /*7:*/
cross1([1,2,3],[4,5,6]);                                            /*8:*/
```

▷ Click here to run this script.
MAXIMA output:

```
[%i2] ishow( w([a],[]) = 'levi_civita([a,b,c],[]) * v([],[b]) * r([],[c]))$
[%t2]
                                b  c
                          w  = v  r  levi_civita
                           a                    a b c
[%i3] ishow( subst([a=1],%))$
[%t3]
                                b  c
                          w  = v  r  levi_civita
                           1                    1 b c
ishow( ev(%, levi_civita))$
[%t4]
                                   b  c       1 2 3
                             w  = v  r  kdelta
                              1                1 b c
ishow( expand( ev(%, kdelta)))$
[%t5]
                                   b  c       2       3
                             w  = v  r  kdelta  kdelta
                              1                b       c
                                b  c       3       2
                             - v  r  kdelta  kdelta
                                           b       c
ishow( contract(%))$
[%t6]
                                      2  3    2  3
                                w  = v  r  - r  v
                                 1
define( cross1(v,r), rhs( ic_convert(c=%)) );
[%o7]
                    cross1(v, r) := w([1], []) =
                   v([], [2]) r([], [3]) - r([], [2]) v([], [3])
[%i8] cross1(v,r) := v[2]*r[3]-r[2]*v[3];
[%o8]
                          cross1(v, r) := v  r  - r  v
                                           2  3    2  3
[%i9] cross1([1,2,3],[4,5,6]);
[%o9]
                                       - 3
```

Comment. In (1:) we translate the defining mathematical tensor term for ω^a into itensor formulation. 2: picks only the first component of it. 3: evaluates function *levi_civita*, which is in 1: only given in nown form (with '..) in order to look at the correct translation. 4: disassembles *levi_civita* into special *kdelta*'s in (%t5), because *levi_civita* is constructed by a generalized *kdelta*, see (%t4). Now 5: contracts the expanded term setting $b \mapsto 2$ and $c \mapsto 3$, look at (%t5). So we now can extract in 6: the formula for the correct first component of *cross* and define in 7: the function *cross1*, we are lokking for. Step 6: is unneccessary, because we can see the right term after step 5: in (%t6).
You may follow the whole process in the screenshot above.

Exercise 33. Calculate the 2nd and 3rd component of the orbital velocity.
Define then the complete orbital velocity $cross(v, r)$.

2nd Second, here is a fully tensorial definition of the cross product.

```
/*-- contributed by V. Toth */
load(itensor)$
load(ctensor)$
dim : 3$
ishow( c([],[k]) = 'levi_civita([],[i,j,k]) * a([i],[]) * b([j],[]))$
ishow( ev(%, levi_civita));                                  /*(1)*/
ishow( expand( ev(%, kdelta)));                             /*(2)*/
ishow( contract(%));                                       /*(3)*/
EQ : ic_convert(%);                                       /*(4)*/
c  : [false,false,false];                                /*(5)*/
kdelta : ident(dim);                                    /*(6)*/
ev(EQ, 'kdelta);                                       /*(7)*/

define(cross(a,b),
           block( local(a,b), a(u,v):=a[u[1]], b(u,v):=b[u[1]], c) ) $
cross([1,2,3],[4,5,6]);
```

Comment. To understand these code lines TOTH recommends: "Watch, what `ic_convert` actually does with an assignment expression. Inspect the output of `ic_convert()` visually, try to make sense of it, understand what it does (in this cases, assigns values to the elements of the array $c[]$), and what it needs (in this case, $'kdelta$ would remain unevaluated had we not assigned to it the value of the identity matrix.)" [27]
Using the system variable %, which references the last executed command, we can follow the process line by line. First of all, $c([],[k]) =$ .. make an array, which catch the three components of the cross product. Line (1) activates the permutation symbol, which was

[27] The original definition for *cross* by V. Toth, which followed after line (7), see (%i8), was:

```
define(cross(a,b),
   block([EQ:ic_convert(%), c:[false,false,false], kdelta:ident(dim)],
   local(a,b), a(u,v):=a[u[1]], b(u,v):=b[u[1]], ev(EQ,'kdelta),c))$
```

I have splittet this long command to make the individual commands citable.

given the line before as a nown and let us inspect the defining formula. (2) expands the activated formula $c([],[k])$ with the help of *kdelta*. Now (3) triggers the summations. The result is converted to an expression saved on variable EQ(uation). The array $c([],[k])$ is now in (5) predefined as a 3-dim vector c and *kdelta* declared as the 3-dim identity matrix.[28]

▷ Click here to run this script.

YAMWI output:

```
[%i7] ishow( contract(%));
```

```
[%t7]
                      k               k               k
                     c  = a  b  kdelta  - b  a  kdelta
                           1  2       3    1  2       3
                                 k             k
                     - a  kdelta  b  + kdelta  a  b
                        1       2  3          1  2  3
                                 k             k
                     + b  kdelta  a  - kdelta  b  a
                        1       2  3          1  2  3

                   c([], [k]) = a([1], []) b([2], [])
                   kdelta([3], [k]) - b([1], []) a([2], [])
                   kdelta([3], [k]) - a([1], [])
                   kdelta([2], [k]) b([3], [])
                   + kdelta([1], [k]) a([2], []) b([3], [])
                   + b([1], []) kdelta([2], [k]) a([3], [])
                   - kdelta([1], [k]) b([2], []) a([3], [])
```

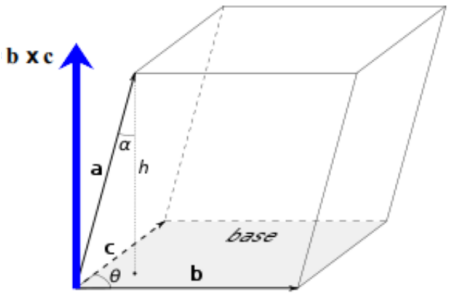
```
[%i8] define(cross(a,b), block([ EQ : ic_convert(%),
                                 c  : [false,false,false],
                                 kdelta : ident(dim) ],
                       local(a,b), a(u,v):=a[u[1]], b(u,v):=b[u[1]],
                       ev(EQ,'kdelta), c ))$
```

```
[%i9] cross([1,2,3],[4,5,6]);
```

```
[%o9]                          [- 3, 6, - 3]
```

[28] Why `c:[false,false,false]`? Why `kdelta:ident(dim)`? Here is the explanation by V. TOTH ▷3

Example 19. The scalar triple product[29] $spat(a,b,c) := a \cdot (b \times c)$ of three vectors **a**, **b** and **c** is defined using the levi-civita symbol and then simplified using 'contract'.



□: the plane, spanned by vectors **b** and **c**.
Figure 19: ↑: the cross product $(b \times c)$
$[\mathbf{a},\mathbf{b},\mathbf{c}]$: the spat product as volume of the feldspar

Task: calculate `spat( [1,2,3], [4,5,6], [7,8,8] )` using `itensor`.

Solution: We use the defining term for function $spat : (\mathbb{R}^3)^3 \to \mathbb{R}^3$

$$spat(\mathbf{a},\mathbf{b},\mathbf{c}) := \sum_{i=1}^{3}\sum_{j=1}^{3}\sum_{k=1}^{3} \varepsilon^{ijk} a_i b_j c_k \tag{9.3}$$

and translate it to itensor language:

$$spat(a,b,c) \overset{Math}{:=} a \cdot (b \times c) \longrightarrow \overset{tensor}{\varepsilon^{ijk}} a_i b_j c_k \longrightarrow \overset{itensor}{\texttt{contract}}$$

1st a half-way `itensor` solution as first step

```
load("itensor")$
ishow( c = levi\_civita([],[i,j,k]) * a([i],[]) * b([j],[]) * c([k],[]))$ /*1:*/
ishow( contract( expand( ev(%, kdelta))))$        /*2:*/
define( spat(a,b,c), rhs( ic\_convert(%)) );     /*3:*/

/* read off by eye from line 3: */
spat(a,b,c) := a[1]*b[2]*c[3] - b[1]*a[2]*c[3]
             - a[1]*c[2]*b[3] + c[1]*a[2]*b[3]
             + b[1]*c[2]*a[3] - c[1]*b[2]*a[3];

spat( [1,2,3], [4,5,6], [7,8,8] );
```

[29] Figure 19 adapted from wikipedia, a(bxc). We use the name 'spat', because it invokes the typical picture of a 'feldspare' aka parallelepiped in your head. In German speaking counties the scalar triple product is therefore often called 'spat product'.

Comment. Line 1: is the translated RHS of spat-formula in itensor language. In (2:) we contract the multiplied out sums and products from the evaluated (with *kdelta*) term in 1:. Line 3: displays the correct defining term for the triple product $spat(a, b, c)$, look at (%t3). Now we *simply read off* the definition of $spat(a, b, c)$ in (%o5) by deleting the blank brackets [ ].

▷ Click here to run this script.

YAMWI output:

```
[%i1] load("itensor")$
[%i2] ishow( levi_civita([],[i,j,k]) * a([i],[]) * b([j],[]) * c([k],[]))$
[%t2]                                i j k
                           kdelta       a  b  c
                                 1 2 3  i  j  k
[%i3] ishow( contract( expand( ev(%, kdelta))))$
[%t3]              a  b  c  - b  a  c  - a  c  b  + c  a  b
                    1  2  3    1  2  3    1  2  3    1  2  3
                   + b  c  a  - c  b  a
                      1  2  3    1  2  3
      define( spat(a,b,c), rhs( ic_convert(c=%)) );
[%o4]                  spat(a, b, c) := a([1], []) b([2], [])
                       c([3], []) - b([1], []) a([2], [])
                       c([3], []) - a([1], []) c([2], [])
                       b([3], []) + c([1], []) a([2], [])
                       b([3], []) + b([1], []) c([2], [])
                       a([3], []) - c([1], []) b([2], [])
                       a([3], [])
[%i5] spat(a,b,c) := a[1]*b[2]*c[3] - b[1]*a[2]*c[3]
                   - a[1]*c[2]*b[3] + c[1]*a[2]*b[3]
                   + b[1]*c[2]*a[3] - c[1]*b[2]*a[3];
[%o5]                  spat(a, b, c) := a  b  c  - b  a  c
                                        1  2  3    1  2  3
                       - a  c  b  + c  a  b  + b  c  a  - c  b  a
                          1  2  3    1  2  3    1  2  3    1  2  3
[%i6] spat( [1,2,3], [4,5,6], [7,8,8] );
[%o6]                                  3
```

2nd We now don't want to read off the solution by eye, but let `i/c/tensor` do the work for us. Therefore, here is a fully tensorial definition of the *spat* product.

```
/*-- this variant is contributed by V. Toth */
load(itensor)$
load(ctensor)$
dim : 3$
ishow( x = levi_civita([],[i,j,k]) * a([i],[]) * b([j],[]) * c([k],[]))$
ishow( contract( ev(%, kdelta, expand)))$
define( spat(a,b,c), rhs( ic_convert( % )))$
```

```
a(u,v):=a[u[1]]$                                                    /* 1: */
b(u,v):=b[u[1]]$
c(u,v):=c[u[1]]$
spat([1,2,3],[4,5,6],[7,8,8]);
```

Comment. Line 1: is crucial. Let's listen to V. TOTH's explanation: "When we end up with number-valued indices, our approach runs into a limitation of `ic_convert()`, which stems from a limitation of *levi_civita* because it is contrary to itensor's design philosophy of _abstract_ indices. These are best used only in interim results in the itensor workflow, not as the final 'product', in which case extra care is needed in their interpretation." [30] [31]

▷ Click here to run this script.

YAMWI output:

```
[%i4]  ishow( levi_civita([],[i,j,k]) * a([i],[]) * b([j],[]) * c([k],[]))$
[%t4]                                  i j k
                              kdelta       a  b  c
                                     1 2 3  i  j  k
[%i5]  ishow( contract( ev(%, kdelta, expand)))$
[%t5]          a  b  c  - b  a  c  - a  c  b  + c  a  b
                1  2  3    1  2  3    1  2  3    1  2  3
               + b  c  a  - c  b  a
                  1  2  3    1  2  3
       define( spat(a,b,c), rhs( ic_convert( x=% )))$
[%i7]  a(u,v):=a[u[1]]$
[%i8]  b(u,v):=b[u[1]]$
[%i9]  c(u,v):=c[u[1]]$
[%i10] spat([1,2,3],[4,5,6],[7,8,8]);
[%o10]                                  3
```

Remark. If you feel that this section closely resembles the one on determinants and cross, you are correct: the *spat* product is merely a geometric variant of the determinant – hence the comparable tensorial representation. Nevertheless, we included this section here to illustrate the use of the `contract`'ion operator through an additional, but familiar example.

[30] In case you want more and detailed explanation ▷1.
[31] Here is an even cleaner version ▷2.

10 Levi-Civita Tensor ϵ

The Levi-Civita tensor ϵ acts like a metric and is often used to define tensorial objects, cf. [31, p. 340] or [32, p.50] for an application in the Nambu mechanics.

10.1 Definition

We follow Grinfeld [7, p.148].
Let Z denote the determinant of the (covariant) metric tensor $g = g_{ij}$, i.e. $Z := det(g)$. Then we define

1. The square root of Z is called the *volume element* V of the metric, i.e.

$$V := \sqrt{det(g)} \tag{10.1}$$

2. The Levi-Civita tensor ϵ_{ijk} resp. E^{ijk} is defined as the volume scaled permutation symbol e_{ijk}, i.e.

$$\epsilon_{ijk} := V \cdot e_{ijk} \tag{10.2}$$

$$\mathrm{E}^{ijk} := \frac{e^{ijk}}{V} \tag{10.3}$$

Remark. We have

LEXICON	*Math*	Maxima tensor
metric tensor g:	g_{ij}	`lg`
inverse of metric tensor g^{-1}:	g^{ij}	`ug`
volume element V:	$\sqrt{det(g)}$	`sqrt(determinant(g))`
Levi-Civita *tensor* ϵ_{ijk}:	$\sqrt{det(g)} \cdot e_{ijk}$	`epsilon`
Levi-Civita *tensor* E^{ijk}:	$1/\sqrt{det(g)} \cdot e^{ijk}$	`Epsilon`

The ad hoc defined terms ϵ and the big Greek letter *Epsilon* E are noted here as nouns.

10.2 Examples

Example 20. The LEVI-CIVITA tensor in an affine coordinate systems in $\mathbb{R}^2$ can be calculated by MAXIMA as follows, cf. [7, p. 148].

```
/* we introduce affine coordinates in R^2 by */
Z1 : [2,0];
Z2 : [cos(%pi/3),sin(%pi/3)];

/* the covariant basis in affine coordinates is */
Z : matrix(Z1,Z2);

/* the GRAM matrix g, i.e. the metric */
g : zeromatrix(2,2)$
for i thru 2 do for j thru 2 do g[i,j] : Z[i].Z[j];

/*-- volume element in affine coordinates */
Z : determinant(g);
V : sqrt(Z);

load(itensor)$
h[i, j] := levi_civita([i,j])$
eij : genmatrix (h, 2,2);

/* -- levi-civita symbol epsilon */
epsilon : V * eij;
Epsilon : 1/V * eij;
```

▷ Click here to run this script.

YAMWI results:

$$Z = 3$$

$$V = 3^{1/2}$$

$$e_{ij} = \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix}$$

$$\varepsilon = \begin{bmatrix} 0 & 3^{1/2} \\ -3^{1/2} & 0 \end{bmatrix}$$

$$\mathrm{E} = \begin{bmatrix} 0 & \frac{1}{3^{1/2}} \\ -\frac{1}{3^{1/2}} & 0 \end{bmatrix}$$

Example 21. The LEVI-CIVITA tensor in polar coordinates in $\mathbb{R}^2$ is calculated by MAXIMA using the results of example 15.

```
/* the metric tensor in polar coordinates */
assume(r>0);
g : zeromatrix(2,2)$
g[1,1] : 1;
g[2,2] : r^2;
g;

/*-- volume element in polar coordinates */
Z : determinant(g);
V : sqrt(Z);

load(itensor)$
h [i, j] := levi_civita([i,j])$
eij : genmatrix (h, 2,2);

/* -- levi-civita symbol epsilon, represented as matrix */
epsilon : V * eij;
Epsilon : 1/V * eij;
```

▷ Click here to run this script.

YAMWI results:

$$Z = r^2$$

$$V = r$$

$$e_{ij} = \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix}$$

$$\varepsilon = \begin{bmatrix} 0 & r \\ -r & 0 \end{bmatrix}$$

$$\mathrm{E} = \begin{bmatrix} 0 & \frac{1}{r} \\ -\frac{1}{r} & 0 \end{bmatrix}$$

Exercise 34. Calculate the LEVI-CIVITA tensor in cylindrical coordinates in $\mathbb{R}^3$ by MAXIMA using the results of example 16.

11 CHRISTOFFEL Tensor Γ^k_{ij}

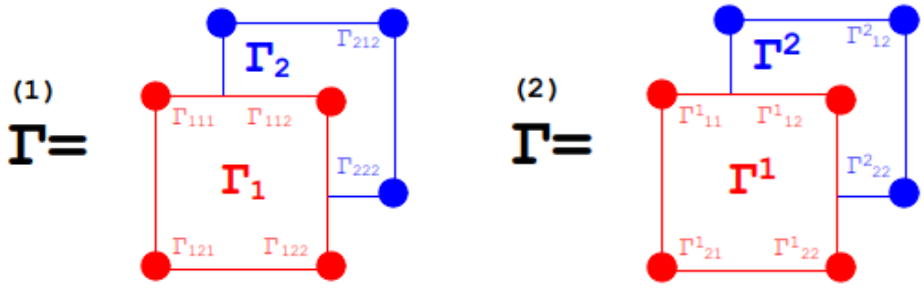


Figure 20: How to imagine the CHRISTOFFEL symbol of first kind $\Gamma^{(1)}$ and the CHRISTOFFEL symbol of second kind $\Gamma^{(2)}$ as $2^3 = 8$ numbers arranged in a tensorial manner in 2 layers. This structure is respected by the output of MAXIMA.

The CHRISTOFFEL tensor (symbol) plays a dominant role in Differential Geometry and General Relativity, where we define the RIEMANN curvature tensor R^a_{bcd}, the RICCI tensor R_{bd} and the EINSTEIN curvature scalar R with the help of the CHRISTOFFEL tensor of 1st and 2nd kind. Then it is possible to calculate curvature variants of surfaces[32] in space. We use the results in [7, p.69] or [32, p.62] to implement the CHRISTOFFEL tensors in MAXIMA.

11.1 Definition

Let $g = (g_{ij})$ be a metric tensor with inverse tensor $ug = g^{ij}$ in the coordinate system (x^1, x^2, x^3) in e.g. $\mathbb{R}^3$. Let $R : U \to \mathbb{R}^n$ be the position vector of the coordinate system.

a. The CHRISTOFFEL *symbol of 1st kind, denoted* $\Gamma_{ij,k}$ where $i, j, k \in \{1, 2\}$ is defined by

$$\Gamma_{ij,k} := \frac{1}{2} \cdot (g_{jk,i} + g_{ik,j} - g_{ij,k}) \stackrel{!}{=} \frac{\partial^2 R}{\partial u_i \partial u_j} \cdot \frac{\partial R}{\partial u_k} \equiv R_{u^i u^j} \bullet R_{u^k} \tag{11.1}$$

where $g_{ij,k} \equiv \frac{\partial g_{ij}}{\partial x^k}$ denotes the partial derivative of g_{ij} with respect to the coordinate x^k.

b. The CHRISTOFFEL *symbol of 2nd kind, denoted* $\Gamma^k{}_{ij}$ is defined by using the inverse metric tensor g^{ij} and memorizing the EINSTEIN summation conventionas

$$\Gamma^k_{ij} := \frac{1}{2} \cdot g^{km} \cdot (g_{mi,j} + g_{mj,i} - g_{ij,m}) \tag{11.2}$$

[32] See e.g. my script in this series [10]

• Decoding the EINSTEIN summation convention and to prepare for the implementation of these formulas in MAXIMA we write equation (11.2) explicit as

$$\Gamma^k_{ij} = \frac{1}{2} \cdot \sum_{m=1}^{2} g^{km} \cdot (g_{mi,j} + g_{mj,i} - g_{ij,m}) \tag{11.3}$$

Implementing formulas a. and b. in MAXIMA should enhance a clear understanding of these constructs.

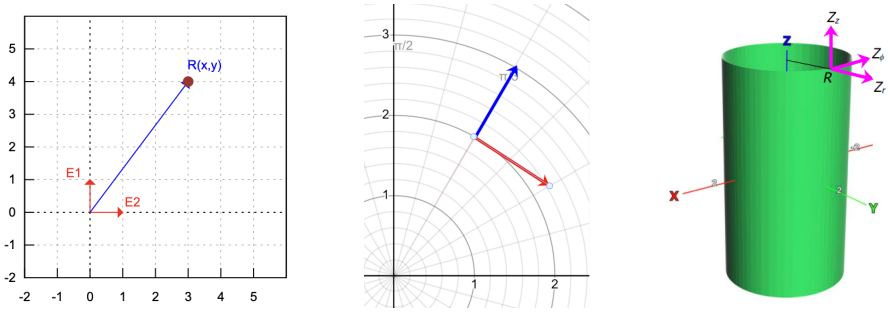
Remark. We have

LEXICON	*Math*	MAXIMA
metric tensor g:	g_{ij}	`g` or `lg`
inverse of metric tensor g^{-1}:	g^{ij}	`ug`
CHRISTOFFEL *tensor of 1st kind*:	Γ^k_{ij}	`christof(lcs); lcs[i,j,k]`
CHRISTOFFEL *tensor of 2nd kind*:	$\Gamma_{i,jk}$	`christof(mcs); mcs[i,j,k]`

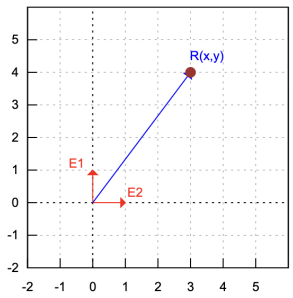
11.2 Examples

We now demonstrate in each example two implementations of the CHRISTOFFEL tensors: one closely oriented on formula **a**.and **b**. and its term structure and the other using clever compact tensor techniques such as contract and index juggling. We test our code in Cartesian, polar and cylindrical coordinates in the plane $\mathbb{R}^2$ resp. in space $\mathbb{R}^3$.

Figure 21:



Left: Cartesian coordinates: not changing, fix basis E_1, E_2.
Middle: polar coordinates: basis E_r, E_φ change from point to point
Right: cylindrical coordinates: basis Z_r, Z_ϕ, Z_z change with the point



Example 22. (The CHRISTOFFEL tensor in *Cartesian coordinates*)
The CHRISTOFFEL tensor in *Cartesian coordinates* in $\mathbb{R}^2$ is calculated with two methods: using plain MAXIMA vs. using tensor package `tensor`.

1st *Calculation of the* CHRISTOFFEL *tensor using standard* MAXIMA
So, let's step through all coefficients of the CHRISTOFFEL symbol's using a for-loop.

- We calculate the CHRISTOFFEL tensor of 1st kind Γ_{ijk} using formula (11.1).

```
"CHRISTOFFEL tensors for the Cartesian plane"$
define( R(u,v) , [u,v]);
U : [u,v];

"metric tensor of the plane"$
gij(i,j) := diff(R(u,v),U[i]) . diff(R(u,v),U[j]);       /* (1) */
g : matrix( [gij(1,1),gij(1,2)], [gij(2,1),gij(2,2)] );  /* (2) */

"CHRISTOFFEL symbol Gamma_ijk of 1st kind using (11.1)"$
ChrI(i,j,k) := diff( diff(R(u,v),U[i]), U[j]) . diff(R(u,v), U[k]); /*(3)*/

ChrI(1,2,1);
ChrI(1,2,2);

lChr : matrix(  [matrix( [ChrI(1,1,1), ChrI(1,1,2)],     /* (4) */
                         [ChrI(1,2,1), ChrI(1,2,2)] )],
                [matrix( [ChrI(2,1,1), ChrI(2,1,2)],
                         [ChrI(2,2,1), ChrI(2,2,2)] )]);

for i:1 thru 2 do                                        /* (5) */
    for j:1 thru 2 do
        for k:1 thru 2 do
            print("ChrI",[i,j,k]," -> ", ChrI(i,j,k));
```

▷ Click here to run this script.

YAMWI output:

[%o7] CHRISTOFFEL symbol Gamma_ijk of 1st kind using (11.1)

```
[%i8] ChrI(i,j,k) := diff( diff(R(u,v),U[i]), U[j]) . diff(R(u,v), U[k]);
```

[%o8] $$\mathrm{ChrI}(i,j,k) := \frac{\partial}{\partial U_j}\frac{\partial}{\partial U_i}R(u,v)\cdot\frac{\partial}{\partial U_k}R(u,v)$$

```
[%i9] /*(3)*/

      ChrI(1,2,1);
```

[%o9] 0

```
[%i10] ChrI(1,2,2);
```

[%o10] 0

```
[%i11] lChr : matrix(  [matrix( [ChrI(1,1,1), ChrI(1,1,2)],     /* (4) */
                                [ChrI(1,2,1), ChrI(1,2,2)] )],
                       [matrix( [ChrI(2,1,1), ChrI(2,1,2)],
                                [ChrI(2,2,1), ChrI(2,2,2)] )]);
```

[%o11] $$\begin{bmatrix}\begin{bmatrix}0 & 0\\ 0 & 0\end{bmatrix} & \begin{bmatrix}0 & 0\\ 0 & 0\end{bmatrix}\end{bmatrix}^T$$

```
[%i12] for i:1 thru 2 do                                        /* (5) */
           for j:1 thru 2 do
               for k:1 thru 2 do
                   print("ChrI", [i,j,k],"->",ChrI(i,j,k));
```

[] ChrI [1,1,1] -> 0

[] ChrI [1,1,2] -> 0

[] ChrI [1,2,1] -> 0

Comment. In (1) we calculate the coefficients of the metric tensor g using the derivations of the position vector R and form the metric's matrix in (2). We respect the EINSTEIN summation convention by explicit summing the three partial derivatives up. In (3) we calculate each individual component of Γ_{ijk}. In (%i9) we display a special value of `ChrI`. All results fully fill the tensor's array, outputted in (%o11). (5) verifies that all components of the Cartesian CHRISTOFFEL tensor tensor Γ vanishes at all points. Later we will argue, that therefore the plane $\mathbb{R}^2$ is 'flat'.

• Now we calculate the CHRISTOFFEL tensor of 2nd kind Γ_{ij}^k using formula (11.3). Again, let's step through all coefficients of the Γ_{ij}^k 's using a for-loop.

```
"CHRISTOFFEL tensor of 2nd kind for the Cartesian plane"$
define( R(u,v) , [u,v]);
U : [u,v];

"metric tensor of the plane"$
gij(i,j) := diff(R(u,v),U[i]) . diff(R(u,v),U[j]);          /* (1) */

g  : matrix( [gij(1,1),gij(1,2)], [gij(2,1),gij(2,2)] );    /* (2) */
gu : invert(g);     /* the inverse of the metric */
```

```
"CHRISTOFFEL symbol Gamma^i_jk of 2nd kind using (11.2)"$
ChrII(k,i,j) := 1/2* sum( gu[k,m]*( diff(g[m,i],U[j]) +        /* (6) */
                        diff(g[m,j],U[i]) - diff(g[i,j],U[m]) ), m,1,2);

ChrII(2,1,2);

uChr : matrix([matrix( [ChrII(1,1,1), ChrII(1,1,2)],           /* (7) */
                       [ChrII(1,2,1), ChrII(1,2,2)] )],
               [matrix( [ChrII(2,1,1), ChrII(2,1,2)],
                       [ChrII(2,2,1), ChrII(2,2,2)] )]);

for i:1 thru 2 do                                              /* (8) */
    for j:1 thru 2 do
        for k:1 thru 2 do
            print("ChrII", [i,j,k]," -> ",ChrII(i,j,k));
```

▷ Click here to run this script.

YAMWI output:

[%o9]

$$\text{ChrII}(k,i,j) := \frac{\text{sum}\left(gu_{k,m}\left(\frac{\partial}{\partial U_i}g_{m,j}+\frac{\partial}{\partial U_j}g_{m,i}-\frac{\partial}{\partial U_m}g_{i,j}\right),m,1,2\right)}{2}$$

```
[%i10] ChrII(2,1,2);
```

[%o10]

$$0$$

```
[%i11] uChr : matrix([matrix( [ChrII(1,1,1), ChrII(1,1,2)],          /* (7) */
                              [ChrII(1,2,1), ChrII(1,2,2)] )],
                     [matrix( [ChrII(2,1,1), ChrII(2,1,2)],
                              [ChrII(2,2,1), ChrII(2,2,2)] )]);
```

[%o11]

$$\begin{bmatrix}\begin{bmatrix}0 & 0\\ 0 & 0\end{bmatrix} & \begin{bmatrix}0 & 0\\ 0 & 0\end{bmatrix}\end{bmatrix}^T$$

```
[%i12] for i:1 thru 2 do                                             /* (8) */
           for j:1 thru 2 do
               for k:1 thru 2 do
                   print("ChrII", [i,j,k]," -> ",ChrII(i,j,k));
```

```
[]  ChrII [1,1,1] -> 0
[]  ChrII [1,1,2] -> 0
```

Comment. In (1) and (2) we calculate the metric g and its inverse gu. In (6) we respect the EINSTEIN summation convention by explicit summing up all three partial derivatives. Again, all components of the Cartesian CHRISTOFFEL tensor tensor $ChrII$ vanishes at all points.

2^{nd} *Calculation of the* CHRISTOFFEL *tensors using* `ctensor`

We now calculate the CHRISTOFFEL symbols using compact tensor techniques with build-in functions of the MAXIMA package ctensor, designed and implemented by V. TOTH. We calculate the CHRISTOFFEL tensors in Cartesian coordinates in $\mathbb{R}^2$, for both kinds in one strike. This demonstrates the power of the tensor package `ctensor` compared to our hand-made calculations in 1^{st}.

```
"CHRISTOFFEL tensors for the Cartesian plane using ctensor."$
load(ctensor)$
ct_coordsys(cartesian2d);                                        /* 1: */
lg;                                                             /* 2: */

"CHRISTOFFEL symbol 1st kind"$
christof(lcs);  /* lower Christoffel symbol */                  /* 3: */
lcs[1,1,2];                                                    /* 4: */

"CHRISTOFFEL symbol 2nd kind"$
christof(mcs);  /* mixed lower and upper indices Christoffel symbol    5: */
mcs[1,1,2];                                                          /* 6: */

/*  only the unique non-zero values of lcs[i,j,k]
    or mcs[i,j,k] will be displayed !           */
```

▷ Click here to run this script.

```
[%i6]  "CHRISTOFFEL symbol 1st kind"$
[%i7]  christof(lcs);
[%o7]                                  done
[%i8]  /* lower christoffel symbol */
       lcs[1,1,2];
[%o8]                                   0
[%i9]  "CHRISTOFFEL symbol 2nd kind"$
[%i10] christof(mcs);
[%o10]                                 done
[%i11] /* mixed (lower and upper indices) christoffel symbol */
       mcs[1,1,2];
[%o11]                                  0
```

Comment. The call 1: sets up the Cartesian coordinate system for 2D plane and calculates

the metric $g = lg$, which is displayed with 2:; this command *sets up a setting*[33] for the use of the package ctensor. All calculated values of the Cartesian metric g are stored in the *ctensor-variable* `lg`. The CHRISTOFFEL symbols are now calculated via a call to the `christof()` function with the arguments `lcs` resp. `mcs`. 3: and 4: print special concrete values of the tensors; please note, that this lines must not be switched, because we must first calculate all values of the tensor before grasping a special value.

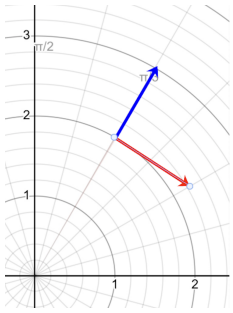
Remark.

1. The CHRISTOFFEL symbols are calculated using only the plane's metric tensor g. Hence this object belongs to the so-called *intrinsic geometry* of the manifold (here: plane).

2. The above presented tensorial version of the calculation of CHRISTOFFEL symbol are most elegant and effective: you only have to set up the necessary requirements. So one may calculate the metric and the derived objects in the same way in higher dimensions, which is essential in General Relativity, see e.g. calculations in the 4D SCHWARZSCHILD space time, cf. OLOFF [19, p. 96 ff] or TOTH [38]. Using user defined `for`-loop constructs to determine the CHRISTOFFEL symbols - which are perhaps more simple, insightful and of a didactic surplus, but more laborious - you must therefore manually specify the considered dimension of the space, the choosen coordinate system and calculate the metric g or g^{-1}

> ♡ This was a long example. But we wanted to present a gentle rise of abstraction in order to fully understand what is going on and to equip the user which a selection of means to be choose from.

[33] Calling the `ctensor` function `csetup();` after the `load` command instead of 1: , the user is led along the whole setup process in an interactive way.



Example 23. (the CHRISTOFFEL tensor in a polar coordinates)
Calculate the CHRISTOFFEL tensor in a polar coordinate system in $\mathbb{R}^2$ in two ways.

1st we calculate the CHRISTOFFEL tensor by for-loops.
Therefore we have to calculate the 'polar' metric first.

```
R(r, theta) := [r*cos(theta),r*sin(theta)];
Z1 : diff(R(r, theta), r);
Z2 : diff(R(r, theta), theta);
Z : matrix(Z1,Z2);

g11 : trigsimp(Z1.Z1);
g12 : Z1.Z2;
g21 : Z2.Z1;
g22 : trigsimp(Z2.Z2);

g : matrix([g11,g12],[g21,g22]);
gu : invert(g);

U : [r, theta];
"CHRISTOFFEL symbol Gamma_ijk of 1st kind using (11.1)"$
ChrI(i,j,k) := diff( diff(R(r, theta),U[i]), U[j]) . diff(R(r, theta), U[k]);

ChrI(1,2,1);
ChrI(1,2,2);

lChr : matrix(  [matrix( [ChrI(1,1,1), ChrI(1,1,2)],
                         [ChrI(1,2,1), ChrI(1,2,2)] )],
                [matrix( [ChrI(2,1,1), ChrI(2,1,2)],
                         [ChrI(2,2,1), ChrI(2,2,2)] )]);

for i:1 thru 2 do
    for j:1 thru 2 do
        for k:1 thru 2 do
            print("ChrI",[i,j,k]," -> ", ChrI(i,j,k));
```

```
"CHRISTOFFEL symbol Gamma^i_jk of 2nd kind using (11.2)"$
ChrII(k,i,j) := 1/2* sum( gu[k,m]*( diff(g[m,i],U[j]) +
                      diff(g[m,j],U[i]) - diff(g[i,j],U[m]) ), m,1,2);

/* exercise for you */
```

▷ Click here to run this script.
YAMWI output:

```
[%i12] "CHRISTOFFEL symbol Gamma_ijk of 1st kind using (11.1)"$
[%i13] ChrI(i,j,k) := diff( diff(R(r, theta),U[i]), U[j]) . diff(R(r, theta), U[k]);
```

[%o13] $$\mathrm{ChrI}(i,j,k) := \frac{\partial}{\partial U_j}\frac{\partial}{\partial U_i}R(r,\vartheta)\cdot\frac{\partial}{\partial U_k}R(r,\vartheta)$$

```
[%i14] /*(3)*/

       ChrI(1,2,1);
```

[%o14] $$0$$

```
[%i15] ChrI(1,2,2);
```

[%o15] $$r(\sin\vartheta)^2 + r(\cos\vartheta)^2$$

```
[%i16] lChr : matrix(  [matrix( [ChrI(1,1,1), ChrI(1,1,2)],     /* (4) */
                                [ChrI(1,2,1), ChrI(1,2,2)] )],
                       [matrix( [ChrI(2,1,1), ChrI(2,1,2)],
                                [ChrI(2,2,1), ChrI(2,2,2)] )]);
```

[%o16] $$\left[\begin{bmatrix} 0 & 0 \\ 0 & r(\sin\vartheta)^2 + r(\cos\vartheta)^2 \end{bmatrix}\quad \begin{bmatrix} 0 & r(\sin\vartheta)^2 + r(\cos\vartheta)^2 \\ -\left(r(\sin\vartheta)^2\right) - r(\cos\vartheta)^2 & 0 \end{bmatrix}\right]^T$$

```
[%i17] for i:1 thru 2 do                                        /* (5) */
           for j:1 thru 2 do
               for k:1 thru 2 do
                   print("ChrI",[i,j,k]," -> ", ChrI(i,j,k));
```

[] ChrI [1,1,1] -> 0

[] ChrI [1,1,2] -> 0

[] ChrI [1,2,1] -> 0

[] ChrI [1,2,2] -> $r(\sin\vartheta)^2 + r(\cos\vartheta)^2$

```
[%i18] "CHRISTOFFEL symbol Gamma^i_jk of 2nd kind using (11.2)"$
[%i19] ChrII(k,i,j) := 1/2* sum( gu[k,m]*( diff(g[m,i],U[j]) +        /* (6) */
                                 diff(g[m,j],U[i]) - diff(g[i,j],U[m]) ), m,1,2);
```

[%o19] $$\mathrm{ChrII}(k,i,j) := \frac{\mathrm{sum}\left(\mathrm{gu}_{k,m}\left(\frac{\partial}{\partial U_i}g_{m,j} + \frac{\partial}{\partial U_j}g_{m,i} - \frac{\partial}{\partial U_m}g_{i,j}\right), m, 1, 2\right)}{2}$$

```
[%i20] ChrII(2,1,2);
```

[%o20] $$\frac{1}{r}$$

```
[%i21] uChr : matrix([matrix( [ChrII(1,1,1), ChrII(1,1,2)],          /* (7) */
                              [ChrII(1,2,1), ChrII(1,2,2)] )],
                     [matrix( [ChrII(2,1,1), ChrII(2,1,2)],
                              [ChrII(2,2,1), ChrII(2,2,2)] )]);
```

[%o21] $$\left[\begin{bmatrix} 0 & 0 \\ 0 & -r \end{bmatrix}\quad \begin{bmatrix} 0 & \frac{1}{r} \\ \frac{1}{r} & 0 \end{bmatrix}\right]^T$$

```
[%i22] for i:1 thru 2 do                                        /* (8) */
           for j:1 thru 2 do
               for k:1 thru 2 do
                   print("ChrII", [i,j,k]," -> ",ChrII(i,j,k));
```

[] ChrII [1,1,1] -> 0

[] ChrII [1,1,2] -> 0

[] ChrII [1,2,1] -> 0

[] ChrII [1,2,2] -> $-r$

Comment. In (%o16) and (%o20) we see that the fully filled CHRISTOFFEL tensor Γ has 3 non-zero values, (%i14) and (%i14) picks two values of both values of $\Gamma = (\Gamma, {}^1\Gamma^2)$, namely $\Gamma^1_{22} = 0$ and $\Gamma^2_{12} = r$.

2st Calculate the CHRISTOFFEL tensor in polar coordinates in $\mathbb{R}^2$ using tensor functions of package `ctensor`.

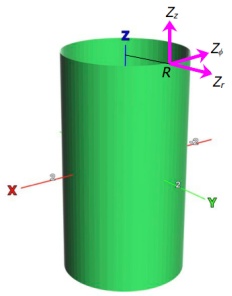
```
"CHRISTOFFEL tensors for the Cartesian plane using ctensor."$
load(ctensor)$
ct_coordsys(polar);                                          /* 1: */
lg;                                                         /* 2: */
ug : invert(lg);

"CHRISTOFFEL symbol 1st kind"$
christof(lcs);  /* lower Christoffel symbol */              /* 3: */
lcs[1,2,2];                                                /* 4: */

"CHRISTOFFEL symbol 2nd kind"$
christof(mcs);  /* mixed lower and upper indices Christoffel symbol    5: */
mcs[1,2,2];                                                        /* 6: */
```

▷ Click here to run this script.

MAXIMA output:

```
[%i6]  "CHRISTOFFEL symbol 1st kind"$
[%i7]  christof(lcs);
[%t7]  (%t7) lcs        = r
              1, 2, 2
[%t8]  (%t8) lcs        = - r
              2, 2, 1
[%o8]   done
[%i9]  /* lower Christoffel symbol */
       lcs[1,2,2];
[%o9]   r
[%i10] "CHRISTOFFEL symbol 2nd kind"$
[%i11] christof(mcs);
[%t11]                        1
       (%t11) mcs         = -
                 1, 2, 2    r
[%t12] (%t12) mcs       = - r
                 2, 2, 1
```

Comment. In (1:) the call to `ct_coordsys` specifies the polar coordinate system with their dimension 2 and the associated metric lg. The call to `christof(lcs)` resp. `christof(mcs)` calculates the values for the CHRISTOFFEL tensor of 1st resp. 2nd kind. We check, that there are only 2 values are different from 0.



Example 24. Calculate the CHRISTOFFEL tensor in cylindrical coordinates in $\mathbb{R}^3$.

1st we calculate the CHRISTOFFEL tensor by for-loops.

Therefore we have to calculate the 'cylindrical' metric first.

```
R(r, theta, z) := [r*cos(theta),r*sin(theta), z];
U : [r,theta,z];
g(i,j) := diff(R(r,theta,z), U[i]) . diff(R(r,theta,z), U[j])$
g : genmatrix (lambda ([i, j], g(i,j)), 3, 3);
g : trigsimp(g) ;
ug : invert(g);

"CHRISTOFFEL symbol Gamma_ijk of 1st kind using (11.1)"$
ChrI(i,j,k) := diff( diff(R(r,theta,z),U[i]), U[j]) . diff(R(r,theta,z), U[k]);
ChrI(1,2,1);
ChrI(1,2,2);
lChr : matrix( [matrix( [ChrI(1,1,1), ChrI(1,1,2), ChrI(1,1,3)],
                        [ChrI(1,2,1), ChrI(1,2,2), ChrI(1,2,3)],
                        [ChrI(1,3,1), ChrI(1,3,2), ChrI(1,3,3)] )],
               [matrix( [ChrI(2,1,1), ChrI(2,1,2), ChrI(2,1,3)],
                        [ChrI(2,2,1), ChrI(2,2,2), ChrI(2,2,3)],
                        [ChrI(2,3,1), ChrI(2,3,2), ChrI(2,3,3)] )],
               [matrix( [ChrI(3,1,1), ChrI(3,1,2), ChrI(3,1,3)],
                        [ChrI(3,2,1), ChrI(3,2,2), ChrI(3,2,3)],
                        [ChrI(3,3,1), ChrI(3,3,2), ChrI(3,3,3)] )] );

for i:1 thru 3 do
    for j:1 thru 3 do
        for k:1 thru 3 do
            print("ChrI",[i,j,k]," -> ", ChrI(i,j,k));

"CHRISTOFFEL symbol Gamma^i_jk of 2nd kind using (11.2)"$
"EXERCISE for the reader."$
```

▷ Click here to run this script.

Comment. Here we have a metric g and a CHRISTOFFEL tensor Γ of 3 dimensions. Therefore the CHRISTOFFEL tensor is a stacked 3-dim array consisting of three 3-by-3 matrices. Only the first two components of Γ, i.e. $\Gamma^1_{\cdot\cdot}$ and $\Gamma^2_{\cdot\cdot}$ has non-zero entries.

YAMWI output:

```
[%i7] "CHRISTOFFEL symbol Gamma_ijk of 1st kind using (11.1)"$
[%i8] ChrI(i,j,k) := diff( diff(R(r,theta,z),U[i]), U[j]) . diff(R(r,theta,z), U[k]);
```

[%o8]

$$ChrI(i,j,k) := \frac{\partial}{\partial U_j}\frac{\partial}{\partial U_i}R(r,\vartheta,z)\cdot\frac{\partial}{\partial U_k}R(r,\vartheta,z)$$

```
[%i9] lChr : matrix( [matrix( [ChrI(1,1,1), ChrI(1,1,2), ChrI(1,1,3)],
                              [ChrI(1,2,1), ChrI(1,2,2), ChrI(1,2,3)],
                              [ChrI(1,3,1), ChrI(1,3,2), ChrI(1,3,3)] )],
                     [matrix( [ChrI(2,1,1), ChrI(2,1,2), ChrI(2,1,3)],
                              [ChrI(2,2,1), ChrI(2,2,2), ChrI(2,2,3)],
                              [ChrI(2,3,1), ChrI(2,3,2), ChrI(2,3,3)] )],
                     [matrix( [ChrI(3,1,1), ChrI(3,1,2), ChrI(3,1,3)],
                              [ChrI(3,2,1), ChrI(3,2,2), ChrI(3,2,3)],
                              [ChrI(3,3,1), ChrI(3,3,2), ChrI(3,3,3)] )]
     );
```

[%o9]

$$\left[\begin{bmatrix}0 & 0 & 0\\ 0 & r(\sin\vartheta)^2+r(\cos\vartheta)^2 & 0\\ 0 & 0 & 0\end{bmatrix}\begin{bmatrix}0 & r(\sin\vartheta)^2+r(\cos\vartheta)^2 & 0\\ -\left(r(\sin\vartheta)^2\right)-r(\cos\vartheta)^2 & 0 & 0\\ 0 & 0 & 0\end{bmatrix}\begin{bmatrix}0 & 0 & 0\\ 0 & 0 & 0\\ 0 & 0 & 0\end{bmatrix}\right]^T$$

2nd Calculate the CHRISTOFFEL tensor in cylindrical coordinates in $\mathbb{R}^3$ using tensor techniques.

```
"CHRISTOFFEL tensors for the cylinder."$
load(ctensor)$
ct_coordsys(polarcylindrical);
lg;
ug : invert(lg);

"CHRISTOFFEL symbol 1st kind"$
christof(lcs);  /* lower Christoffel symbol */

"CHRISTOFFEL symbol 2nd kind"$
christof(mcs);  /* mixed lower and upper indices Christoffel symbol   */
```

▷ Click here to run this script.

YAMWI output:

```
[%i5]  ug : invert(lg);
```

[%o5]
$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & \frac{1}{r^2} & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

```
[%i6]  "CHRISTOFFEL symbol 1st kind"$
[%i7]  christof(lcs);
```

[%t7] $lcs_{1,2,2} = r$

[%t8] $lcs_{2,2,1} = -r$

[%o8] done

```
[%i9]  /* lower Christoffel symbol */

       "CHRISTOFFEL symbol 2nd kind"$
[%i10] christof(mcs);
```

[%t10] $mcs_{1,2,2} = \frac{1}{r}$

[%t11] $mcs_{2,2,1} = -r$

[%o11] done

Comment. We repeat the same tensorial calculation as in the examples before. The only difference is in the call to `ct_coordsys()`, where we choose `cartesian2d`, `polar` resp. `polarcylindrical` to run the different examples. In contrast to the 'classic' solution using for-loops the higher dimension of Γ is automatically detected, no special dimension marker like '$dim = 3$' is necessary: the dimension of CHRISTOFFEL tensor $\Gamma^{\cdot}_{\cdot\cdot}$ is taken from the specified coordinate system name.

Exercise 35. Calculate the CHRISTOFFEL tensor Γ_{abc} of 1st kind for the last example using the classical way via for-loops. Verify your calculation with a check w.r.t. the tensor method.

♡

This ends our journey to the first elements of Tensor Calculus. See you again doing the first steps into the *Elementary Differential Geometry of Surfaces* at [10].

♡

Here are some recommendation for further study of `a/c/itensor`:
TOTH, PIPIN & AL. [39], VOLINSKI [43] or TOTH [38].

11.3 Riemann Curvature Tensor $R_{ijk\ell}$: a Preview

The CHRISTOFFEL symbols play a prominent role in Differential Geometry and General Relativity, where we define the RIEMANN curvature tensor $R^a{}_{bcd}$, the RICCI tensor R_{bd} and the RIEMANN curvature scalar R with the help of the CHRISTOFFEL symbols of 1^{st} and 2^{nd} kind. One needs the RIEMANN curvature tensor $R^a{}_{bcd}$ e.g. to formulate the famous GAUSSian *theorema egregium*.

Definition.
The RIEMANN *curvature tensor of* 1^{st} *kind, denoted* R_{ijkl}: $U \to \mathbb{R}$ is defined by

$$R_{ijkl} := \partial_k \Gamma_{jli} - \partial_l \Gamma_{jki} + \Gamma_{ilr} \cdot \Gamma^r_{jk} - \Gamma_{ikr} \cdot \Gamma^r_{jl} \tag{11.4}$$

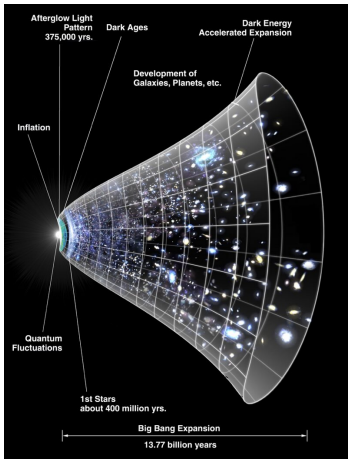
and the RIEMANN *curvature tensor of* 2^{nd} *kind, denoted* R^i_{jkl} is defined by

$$R^i{}_{jkl} := \partial_k \Gamma^i_{jl} - \partial_l \Gamma^i_{jk} + \Gamma^r_{jl} \cdot \Gamma^i_{rk} - \Gamma^r_{jk} \cdot \Gamma^i_{rl} \tag{11.5}$$

Remark.
1. We remark that the latin indices i, j, k, l used in the previous equations do not imply that we are in 3D ambient space, but are meant here to denote the indexed parameters of the RIEMANN curvature tensor of the manifold.
2. The RIEMANN curvature tensor R of type $(4, 0)$ resp. $(3, 1)$ has $2^4 = 16$ coefficients arranged in 4 2-by-2 matrices. *Recall*: in MAXIMA we encode the tensorial position of the first index as 'upper index' with 'u' and 'lower index' with 'l' in the notations of the CHRISTOFFEL symbols Γ^*_* and the RIEMANN curvature tensors R^*_*. This convention is oriented to V. TODT's itensor package for MAXIMA, which is very useful in this context.
3. We have

LEXICON	*Math*	MAXIMA(this book)	ctensor (package)
RIEMANN *curvature tensor of* 1^{st} *kind*:	R_{ijkl}	lR or RI	lriemann or lriem
RIEMANN *curvature tensor of* 2^{nd} *kind*:	$R^i{}_{jkl}$	uR or RII	uriemann or uriem

To see the CHRISTOFFEL tensor and the curvature tensor $R^a{}_{bcd}$ at work, look e.g. at [10].



A Endnotes

Here are some valuable hints by V. Toth to some subtile features of Maxima and especially w.r.t. the tensor packages `itensor` and `ctensor` given in private communications.

1. (Private communication of V. Toth, 5.5.26:)

   Because a([3],[]) is not a regular object in itensor, `lc_convert()` does not know how to convert it. So it keeps it intact, in this form when it returns the expression

   $x : a([1],[]) * b([2],[]) * c([3],[]) - b([1],[]) * a([2],[]) * c([3],[]) - a([1],[]) * c([2],[]) * b([3],[]) + c([1],[]) * a([2],[]) * b([3],[]) + b([1],[]) * c([2],[]) * a([3],[\,]) - c([1],[]) * b([2],[]) * a([3],[])$

   To Maxima outside of itensor, a([3],[]) is just a call to a function named 'a', with two parameters: a 1-element list '[3]', and an empty list '[]'.

   So let us actually declare a function with two parameters, and then extract the one thing that matters to us, the index that is the element of the first list:

   $a(u, v) := a[u[1]];$

   When a() is called, u = [3], v = []; we discard v, and we use u[1] = 3.

   We then make use of the fact that 'a' can simultaneously be the name of a function and an object with list attributes. Mind you, this was not truly necessary. I could have written

   ```
   load(itensor)$
   load(ctensor)$
   dim:3$
   ishow( levi_civita([],[i,j,k]) * a([i],[]) * b([j],[]) * c([k],[]))$
   ishow(contract(ev(%,kdelta,expand)))$
   define(spat(p,q,r),rhs(ic_convert(x=%)))$
   a(u,v):=p[u[1]]$
   b(u,v):=q[u[1]]$
   c(u,v):=r[u[1]]$
   spat([1,2,3],[4,5,6],[7,8,8]);
   ```

   and achieved the exact same result.

   Again, the point is, the function declarations allow Maxima to "make sense" of 'a([3],[])' outside of itensor.

2. (Private communication of V. Toth, 6.5.26:)

   Here's an even cleaner version that keeps the required function definitions local to the *spat*() function, so no global name leakage:

   ```
   load(itensor)$
   load(ctensor)$
   dim:3$
   ishow( levi_civita([],[i,j,k]) * a([i],[]) * b([j],[]) * c([k],[]))$
   ishow( contract( ev(%, kdelta, expand)))$
   define(spat(a,b,c), block( [EQ:rhs(ic_convert(x=%))]
                       local(a,b,c),
                       a(u,v):=a[u[1]],
                       b(u,v):=b[u[1]],
                       c(u,v):=c[u[1]],
                       ev(EQ)))$

   spat([1,2,3],[4,5,6],[7,8,8]);
   ```

3. (Private communication from V. Toth, 7.5.26:)

   1. The initialization of c[]. This is basic Maxima semantics. Look:

```
(%i1) c[1]:111;
(%o1)          111
(%i2) c[2]:222;
(%o2)          222
(%i3) c[3]:333;
(%o3)          333
(%i4) c;
(%o4)          c
```

   Oops. What happened? Well, Maxima created a "hashed array". It is very useful; e.g., I could continue with

```
(%o4)          c
(%i5) c[123123123]:-1;
(%o5)          - 1
(%i6) c[x]:-y;
(%o6)          - y
(%i7) c[1];
(%o7)          111
(%i8) c[123123123];
(%o8)          - 1
(%i9) c[x];
(%o9)          - y
```

   and NOT end up with an array of hundreds of millions of uninitialized elements consuming memory. But sometimes this is not what we want. So instead, I want to pre-initialize *c* as a regular array. But to avoid subtle errors, I do not want to initialize it as an array of numbers. If I just did a 'c:[0,0,0]', how could I tell the difference between a program that failed vs. a program that calculated a manifestly zero cross product? Hence my use of 'c:[false,false,false]'. That way, when the end result is '[0,0,0]', I know it's calculated, not a result of a programming error leaving the array untouched.

```
(%i1) c:[false,false,false];
(%o1)          [false, false, false]
(%i2) c[1]:111;c[2]:222;c[3]:333;
(%o2)          111
(%o3)          222
(%o4)          333
(%i5) c;
(%o5)          [111, 222, 333]
```

   2. The initialization of *kdelta*.

   ic_convert does not know anything about kdelta. If we end up with unresolved kdelta terms in an ic_convert expression, we need to tell Maxima that kdelta is really a synonym for the identity matrix.

   3. my point about inspecting ic_convert.

```
(%i1) display2d : false$  load(itensor)$
(%i3) ic_convert(a([],[])=b([i],[])*c([],[i]));
(%o3) a:sum(c[i]*b[i],i,1,dim)
```

```
(%i4) rhs(%);
(%o4) sum(c[i]*b[i],i,1,dim)

(%i5) ic_convert(a([k],[])=b([k,i],[])*c([],[i]));
(%o5) for k thru dim do a[k]:sum(c[i]*b[k,i],i,1,dim)
(%i6) rhs(%);
(%o6) 0
```

See? In the first case, ic_convert was evaluating a scalar-valued assignment, and generated in turn a simple assignment expression in Maxima. One that actually has a right-hand side. In the second case, the expression being evaluated was vector-valued; ic_convert therefore created a loop instruction in Maxima. That has no rhs of course.

4. (Private communication from V. Toth, 9.5.26:)
   For starters,

   ```
   c([],[k]) = 'levi_civita([],[i,j,k]) * a([i],[]) [...]
   ```

   is *NOT* an assignment expression, so nothing is "stored" in c. Also, itensor uses symbols without defining them (that's one of its strengths): the only thing that matters is the arrangement of indices.
   Here:

   ```
   (%i1) display2d:false$
   (%i2) a[1]:123$
   (%i3) a[2]:234$
   (%i4) a;
   (%o4) a

   (%i5) b:[stuff,stuff]$
   (%i6) b;
   (%o6) [stuff,stuff]
   (%i7) b[1]:123;
   (%o7) 123
   (%i8) b[2]:234;
   (%o8) 234
   (%i9) b;
   (%o9) [123,234]
   ```

   When you use ‘a[1]:123‘ on a symbol, ‘a‘, that is uninitialized, Maxima automatically creates a hashed array and associates it with ‘a‘. From then on, ‘[]‘ references elements of that hashed array. To **PREVENT** this, I pre-initialized ‘b‘ to a regular list/matrix, so that ‘[]‘ references the elements of that array on assignment, instead of causing the creation of a hashed array.
   Note also this, to highlight the difference:

   ```
   (%i12) a[123123123]:more_stuff$
   (%i13) a[123123123];
   (%o13) more_stuff
   (%i14) a[123123124];
   (%o14) a[123123124]
   (%i15) b[123123123]:other_stuff;

   assignment: matrix row index 123123123 out of range.
    -- an error. To debug this try: debugmode(true);
   ```

B grad-div-curl in different coordinate systems

To use formula's for the vector analysis 'operators' *grad, div, curl (=rot), laplacian* it is recommended to remember and work with general coordinates for the definitions and afterwards specify the concrete coordinate system instead of memorize many (>= 12) special formula's for special coordinate systems e.g. *cartesian, cylindrical, spherical,* ..see (1')..(4'), $(1^C)..(4^C)$ and $(1^S)..(4^S)$.

Theorem [**Vector Analysis in $\mathbb{R}^3$ with general coordinates**]
Let $u = (u_1, u_2, u_3)$ be a general ('curvilinear') orthogonal coordinate system.
Let $\mathbf{F} = (F_1, F_2, F_3)\colon D \subset \mathbb{R}^3 \to \mathbb{R}^3$ be a C^1 "vector" function with coordinates u.
Let $f(u_1, u_2, u_3)\colon D \subset \mathbb{R}^3 \to \mathbb{R}$ be a C^1 "scalar" function with coordinates u.
Let $R\colon (x, y, z) \mapsto (u_1, u_2, u_3)$ be a transformation between (x, y, z) and (u_1, u_2, u_3).
Let $U := (\frac{\partial R}{\partial u}) = (U_1, U_2, U_3)$ be the *Jacobian* matrix of R with columns U_i.
Let $h = (h_1, h_2, h_3)$ with $h_i := \sqrt{U_i \bullet U_i}$ be the *scale factors* of the transformation R.
Let $\boldsymbol{e_i} := \frac{1}{h_i} U_i = \frac{1}{h_i} \frac{\partial R}{\partial u_i}$ be the unit vector in direction U_i.

Then we define the 'symbolic operators' $\nabla f, \nabla \cdot \mathbf{F}, \nabla \times \mathbf{F}, \Delta f$ alias *grad, div, curl (=rot), laplacian* through the following explicit formular's on the right hand side:

$$\nabla f(u_1, u_2, u_3) := \operatorname{grad}(f(u_1, u_2, u_3)) := \frac{1}{h_1}\frac{\partial f}{\partial u_1}\boldsymbol{e_1} + \frac{1}{h_2}\frac{\partial f}{\partial u_2}\boldsymbol{e_2} + \frac{1}{h_3}\frac{\partial f}{\partial u_3}\boldsymbol{e_3} \tag{B.1}$$

$$\nabla \cdot \mathbf{F} := \operatorname{div} \mathbf{F} := \frac{1}{h_1 h_2 h_3}\left[\frac{\partial}{\partial u_1}(h_2 h_3 F_1) + \frac{\partial}{\partial u_2}(h_3 h_1 F_2) + \frac{\partial}{\partial u_3}(h_1 h_2 F_3)\right] \tag{B.2}$$

$$\nabla \times \mathbf{F} := \operatorname{curl} \mathbf{F} := \frac{1}{h_1 h_2 h_3}\begin{vmatrix} h_1\boldsymbol{e_1} & h_2\boldsymbol{e_2} & h_3\boldsymbol{e_3} \\ \dfrac{\partial}{\partial u_1} & \dfrac{\partial}{\partial u_2} & \dfrac{\partial}{\partial u_3} \\ h_1 F_1 & h_2 F_2 & h_3 F_3 \end{vmatrix} \tag{B.3}$$

$$:= \frac{1}{h_2 h_3}\left(\frac{\partial(h_3 F_3)}{\partial u_2} - \frac{\partial(h_2 F_2)}{\partial u_3}\right) + \frac{1}{h_3 h_1}\left(\frac{\partial(h_1 F_1)}{\partial u_3} - \frac{\partial(h_3 F_3)}{\partial u_1}\right) + \frac{1}{h_1 h_2}\left(\frac{\partial(h_2 F_2)}{\partial u_1} - \frac{\partial(h_1 F_1)}{\partial u_2}\right)$$

$$\begin{aligned} \Delta f &:= \text{Laplacian}(f(u_1, u_2, u_3)) \\ &:= \frac{1}{h_1 h_2 h_3}\left[\frac{\partial}{\partial u_1}\left(\frac{h_2 h_3}{h_1}\frac{\partial f}{\partial u_1}\right) + \frac{\partial}{\partial u_2}\left(\frac{h_3 h_1}{h_2}\frac{\partial f}{\partial u_2}\right) + \frac{\partial}{\partial u_3}\left(\frac{h_1 h_2}{h_3}\frac{\partial f}{\partial u_3}\right)\right] \end{aligned} \tag{B.4}$$

Verify using MAXIMA:

1. Get the usual standard expressions for the 4 operators using **rectangular cartesian coordinates** (x, y, z) for (u_1, u_2, u_3) in the formular's (1)..(4):

   (1') $\nabla f = \text{grad}(f) = \frac{\partial f}{\partial x}\mathbf{i} + \frac{\partial f}{\partial y}\mathbf{j} + \frac{\partial f}{\partial z}\mathbf{k} \equiv (\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z})$

   (2') $\nabla \cdot \mathbf{F} = \text{div}\,\mathbf{F} = \frac{\partial F_1}{\partial x} + \frac{\partial F_2}{\partial y} + \frac{\partial F_3}{\partial z}$

   (3') $\nabla \times \mathbf{F} = \text{curl}(F) = (F_{3,y} - F_{2,z})\mathbf{i} + (F_{1,z} - F_{3,x})\mathbf{j} + (F_{2,x} - F_{1,y})\mathbf{k}$ $\square$[34]

   $= \left(\frac{\partial F_3}{\partial y} - \frac{\partial F_2}{\partial z}\right)\mathbf{i} + \left(\frac{\partial F_1}{\partial z} - \frac{\partial F_3}{\partial x}\right)\mathbf{j} + \left(\frac{\partial F_2}{\partial x} - \frac{\partial F_1}{\partial y}\right)\mathbf{k}$

   (4') $\Delta f \equiv \nabla^2 f \equiv \text{Laplacian}(f) := \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2} + \frac{\partial^2 f}{\partial z^2}$

   a. If $f(x, y, z) = x^2yz^3$ and $\mathbf{F} = xz\mathbf{i} - y^2\mathbf{j} + 2x^2y\mathbf{k}$ find $\nabla f, \nabla\cdot\mathbf{F}, \nabla\times\mathbf{F}, div(f\mathbf{F}), curl(f\mathbf{F}), \Delta f$.

   b. Check 1a. by hand.

2. Get the following expressions for the 4 operators using **cylindrical coordinates** (r, φ, z) for (u_1, u_2, u_3) in the formular's (1)..(4):

   (1^C) $\nabla f = \text{grad}(f) = f_{,r}\mathbf{e}_r + \frac{f_{,\varphi}}{r}\mathbf{e}_\varphi + f_{,z}\mathbf{e}_z$

   $\nabla f(r, \varphi, z) = \frac{\partial f}{\partial r}\mathbf{e}_r + \frac{1}{r}\frac{\partial f}{\partial \varphi}\mathbf{e}_\varphi + \frac{\partial f}{\partial z}\mathbf{e}_z$

   (2^C) $\nabla \cdot \mathbf{F} = \text{div}\,\mathbf{F} = \frac{1}{r}\frac{\partial}{\partial r}(rF_r) + \frac{1}{r}F_{\varphi,\varphi} + F_{z,z}$

   $= \frac{1}{r}\frac{\partial}{\partial r}(rF_r) + \frac{1}{r}\frac{\partial F_\varphi}{\partial \varphi} + \frac{\partial F_z}{\partial z}$

   (3^C) $\nabla \times \mathbf{F} = \left[\frac{1}{r}\frac{\partial F_z}{\partial \varphi} - \frac{\partial F_\varphi}{\partial z}\right]\mathbf{e}_r + \left[\frac{\partial F_r}{\partial z} - \frac{\partial F_z}{\partial r}\right]\mathbf{e}_\varphi + \frac{1}{r}\left[\frac{\partial}{\partial r}(r \cdot F_\varphi) - \frac{\partial F_r}{\partial \varphi}\right]\mathbf{e}_z$

   (4^C) $\Delta f = \nabla^2 f = \text{Laplacian}(f) = \frac{f_{,r}}{r} + f_{,rr} + \frac{f_{,\varphi\varphi}}{r^2} + f_{,zz}$

   a. If $f(r, \varphi, z) = r^2z^3$ and $\mathbf{F} = z\mathbf{e}_r - \varphi^2\mathbf{e}_\varphi + 2r^2z\mathbf{e}_z$
   find $\nabla f, \nabla \cdot \mathbf{F}, \nabla \times \mathbf{F}, div(f\mathbf{F}), curl(f\mathbf{F}), \Delta f$.

   b. Check 2a. by hand.

3. Get the following expressions for the 4 operators using **spherical coordinates** (r, θ, φ) for (u_1, u_2, u_3) in the formular's (1)..(4):

   (1^S) $\nabla f = \text{grad}(f) = f_{,r}\mathbf{e}_r + \frac{f_{,\theta}}{r}\mathbf{e}_\theta + \frac{f_{,\varphi}}{r\sin(\theta)}\mathbf{e}_\varphi$

   $\nabla f(r, \theta, \varphi) = \frac{\partial f}{\partial r}\mathbf{e}_r + \frac{1}{r}\frac{\partial f}{\partial \theta}\mathbf{e}_\theta + \frac{1}{r\sin\theta}\frac{\partial f}{\partial \varphi}\mathbf{e}_\varphi$

   (2^S) $\nabla \cdot \mathbf{F} = \text{div}\,\mathbf{F} = \frac{1}{r^2}\frac{\partial}{\partial r}(r^2F_r) + \frac{1}{r\sin\theta}\frac{\partial}{\partial \theta}(\sin\theta\, F_\theta) + \frac{1}{r\sin\theta}\frac{\partial F_\varphi}{\partial \varphi}$

   (3^S) $\nabla\times\mathbf{A} = \frac{1}{r\sin\theta}\left[\frac{\partial}{\partial \theta}(F_\varphi \sin\theta) - \frac{\partial F_\theta}{\partial \varphi}\right]\mathbf{e}_r + \frac{1}{r}\left[\frac{1}{\sin\theta}\frac{\partial F_r}{\partial \varphi} - \frac{\partial}{\partial r}(rF_\varphi)\right]\mathbf{e}_\theta + \frac{1}{r}\left[\frac{\partial}{\partial r}(rF_\theta) - \frac{\partial F_r}{\partial \theta}\right]\mathbf{e}_\varphi$

   (4^S) $\Delta f = \frac{1}{r^2}\frac{\partial}{\partial r}\left(r^2\frac{\partial f}{\partial r}\right) + \frac{1}{r^2\sin(\theta)}\frac{\partial}{\partial \theta}\left(\sin(\theta)\frac{\partial f}{\partial \theta}\right) + \frac{1}{r^2\sin^2(\theta)}\frac{\partial^2 f}{\partial \varphi^2}$

[34] this is a usual compact short notation for the partial derivative: $\frac{\partial F_3}{\partial y} \equiv F_{3,y}$

$$= \frac{1}{r}\frac{\partial^2}{\partial r^2}(rf) + \frac{1}{r^2\sin\theta}\frac{\partial}{\partial\theta}\left(\sin\theta\frac{\partial f}{\partial\theta}\right) + \frac{1}{r^2\sin^2\theta}\frac{\partial^2 f}{\partial\varphi^2}$$

a. If $f(r,\theta,\varphi) = \frac{1}{2}(2\cos\theta + 3\sin^2\theta\cos\varphi)$ and $\mathbf{F} = \mathbf{e}_r + \frac{2}{\theta}\mathbf{e}_\theta + \varphi r\sin(\theta)\mathbf{e}_\varphi$ find $\nabla f, \nabla\cdot\mathbf{F}, \nabla\times\mathbf{F}, div(f\mathbf{F}), curl(f\mathbf{F}), \Delta f$ in *spherical coordinates* .

b. Check 3a. by hand.

4. Let $\mathbf{F}(x,y,z) = z\mathbf{e}_1 - 2x\mathbf{e}_2 + y\mathbf{e}_3 = (z,-2x,y)$. Write F in cylindrical coordinates. Then calculate $\nabla\cdot\mathbf{F}$ and $\nabla\times\mathbf{F}$ in *cylindrical coordinates*.

5. Derive formular's for $\nabla f, \nabla\cdot\mathbf{F}, \nabla\times\mathbf{F}, \Delta f$ in *parabolic coordinates*, i.e $(x,y,z) = (\frac{1}{2}(u_1^2 - u_2^2), u_1u_2, u_3)$.

a. If $f(u_1,u_2,u_3) = u_1^2u_3^3$ and $\mathbf{F} = u_3\mathbf{e}_1 - u_2^2\mathbf{e}_2 + 2u_1^2u_2\mathbf{e}_3$ find $\nabla f, \nabla\cdot\mathbf{F}, \nabla\times\mathbf{F}, div(f\mathbf{F}), curl(f\mathbf{F}), \Delta f$ in general 'curvilinear' coordinates..

b. Check 3a. by hand.

6. Get expressions for the operators $\nabla f, \nabla\cdot\mathbf{F}, \nabla\times\mathbf{F}, \Delta f$ in **polar coordinates** in $\mathbb{R}^2$.

7. Poof the formular's (1)..(4). See: [29, p. 137], [33, p. 62 ff].

   (1) Grad: see [42, p. 22], [29, p. 148], [33, p. 64], [17, §6.9]

   (2) Divergence: [42, p. 30], [29, p. 150], [33, p. 64], [17, §6.10]

   (3) Curl: [42, p. 31], [29, p. 150], [17, §6.11]

   (4) Laplacian: [42, p. 23], [29, p. 151] , [17, §6.12]

C Bibliography

References

[1] Banchoff, T. F. & Lovett, S. (^{3}2022): *Differential Geometry of Curves and Surfaces.* London: Chapman and Hall.

[2] Bishop, R. L. & Goldberg, S. I. (1980): *Tensor Analysis on Manifolds.* New York: Dover Publications.

[3] Brehmer, S. & Haar, H. (1973): *Differentialformen und Vektoranalysis.* Berlin: VEB Deutscher Verlag der Wissenschaften.

[4] Collier, P. (2021): *A Beginner's Guide to Differential Forms.* Incomprehensible Books.

[5] Collier, P. (^{3}2021): *A Most Incomprehensible Thing.* Incomprehensible Books.

[6] Fleisch, D. (2012): *A Student's Guide to Vectors and Tensors.* Cambridge UK: Cambridge University Press.

[7] Grinfeld, P. (2013): *Introduction to Tensor Analysis and the Calculus of Moving Surfaces.* New York: Springer.

[8] Kaiser, D. (20??): *Maxima Handbuch.*
url: `https://maxima.sourceforge.io/docs/manual/de/maxima_111.html`

[9] Kreyszig, E. (^{2}1968): *Differentialgeometrie.* Leipzig: Akademische Verlagsgesellschaft Geest & Portig.

[10] Lindner, W. (2025): *Classical Differential Geometry of Surfaces with* `Maxima`.
url: `https://lindnerdrwg.github.io/DiffGeo-of-Surfaces-Maxima.pdf`

[11] Lindner, W. (2020): *Didactical APOS theory and other papers.*
url: `https://lindnerdrwg.github.io/`

[12] Lindner, W. (2020): *Linear Algebra interactiv (LAi.5): Determinants.*
url: `https://lindnerdrwg.github.io/LAi5-Determinants.pdf`

[13] Lindner, W. (2020): *Linear Algebra interactiv (LAi.3): Singular Systems and Analytical Geometry.*
url: `https://lindnerdrwg.github.io/LAi3-Singular-Systems-Analytical-Geometry.pdf`

[14] Lipschutz, M. M. (1969): *Differential Geometry.* New York: McGraw-Hill

[15] Lovelock, D. & Rund, H. (1989): *Tensors, Differential Forms and Variational Principles.* New York: Dover Publications.

[16] Marsden, J. & Weinstein, A. ([2]1985): *Calculus I, II, III.* New York: Springer.

[17] Murray, D. (2018): *Vector Operators.*
`https://www.lehman.edu/faculty/anchordoqui/VC-3.pdf`

[18] Needham, T. (2021): *Visual Differential Geometry and Forms.*
Princeton: Princeton University Press.

[19] Oloff, R. ([6]2018): *Geometrie der Raumzeit.* [engl.: Geometry of Spacetime.]
Berlin: Springer Spectrum.

[20] Reckziegel, H. et al. (1998): *Elementare Differentialgeometrie mit Maple.*
Braunschweig: Vieweg.

[21] Rejbrand, A. (2023): *The Rejbrand Encyclopaedia of Curves and Surfaces.*
url: `https://trecs.se/surfaces.php`

[22] Roolfs, G. (2026): *Trägheitsmoment.*
url: `http://groolfs.de/Verschiedenespdf/Traegheitsmoment.pdf`

[23] Roolfs, G. (2026): *Was ist ein Tensor?*
url: `http://groolfs.de/Verschiedenespdf/Tensor2.pdf`

[24] Roolfs, G. (2026): *Tensoren.*
url: `http://groolfs.de/Verschiedenespdf/Tensor.pdf`

[25] Schultz-Piszachich, W. (1977): *Tensoralgebra und -analysis.*
Leipzig: BSB B.G. Teubner.

[26] Snygg, J. (2001): *A New Approach to Differential Geometry using Clifford's Geometric Algebra.* New York: Springer-Birkhäuser.

[27] Sochi, T. (2017): *Tensor Calculus Made Simple.*
?: CreativeSpace. ISBN 9781541013636.

[28] Sochi, T. (2017): *Principles of Tensor Calculus.*
?: CreativeSpace. ISBN 9781974401390.

[29] Spiegel, M.R. (1959): *Vector Analysis and an Introduction to Tensor Analysis.*
New York: McGraw-Hill.

[30] Spiegel, M. R. (1963): *Advanced Calculus.*
New York: McGraw-Hill (Schaums' Outline Series)

[31] Steeb, W.-H. (1991): *Kronecker Product of Matrices and Applications.* Mannheim: Bibliographisches Institut.

[32] Steeb, W.-H., Hardy, Y. (2019): *Kronecker Product.* In: Matrix Calculus, Kronecker Product and Tensor Product.
url: https://www.worldscientific.com/doi/pdf/10.1142/9789811202520_0002

[33] Tong, A.(2020): *Vector Analysis.*
url: https://www.damtp.cam.ac.uk/user/tong/vc/vc3.pdf

[34] Toth, V. T. (2004): *On tensors and their matrix representations.*
url: https://www.vttoth.com/CMS/physics-notes/139

[35] Toth, V. T. (2004): *Vector spaces and metric manifolds.*
url: https://www.vttoth.com/CMS/physics-notes/394

[36] Toth, V. T. (2004): *Maxima - ctensor package functions.*
url: https://maxima.sourceforge.io/docs/manual/maxima_128.html

[37] Toth, V. T. (2004): *Tensor manipulation in GPL Maxima.*
url: https://arxiv.org/abs/cs/0503073

[38] Toth, V. (2008): *The Maxima computer algebra system*
url: https://www.vttoth.com/CMS/projects/61-the-maxima-computer-algebra-system

[39] Toth, V. (2005): *Indicial and Component Tensor Manipulation Demos.*
url: https://www.vttoth.com/DOWNLOAD/maxima_tensor_demo_20080321.txt

[40] Toth, V. T. (2023): *Field theory with the Maxima computer algebra system.*
url: https://arxiv.org/pdf/2308.09837

[41] Villate, D. (2025): *A Maxima tutorial.*
url: https://maxima.st/test.html

[42] Vvedensky, D.(1994): *Partial Differential Equations with* Mathematica.
Cambridge: Addison-Wesley.

[43] Volinski, D. (2025): *Courses.*
url: https://github.com/danielvolinski/Courses

[44] Weigt, G. (2026): Maxima *Homepage.*
url: https://georgeweigt.github.io

[45] WikipediA: Kronecker Product.
url: https://en.wikipedia.org/wiki/Kronecker_product

[46] WikipediA: Levi-Civita-Symbol.
url: https://de.wikipedia.org/wiki/Levi-Civita-Symbol

[47] WikipediA: Metric Tensor.
url: https://en.wikipedia.org/wiki/Metric_tensor

[48] WikipediA: Outer Product.
url: https://en.wikipedia.org/wiki/Outer_product

[49] WikipediA: Tensor Contraction.
url: https://en.wikipedia.org/wiki/Tensor_contraction

[50] WikipediA: Polar coordinate system.
url: https://en.wikipedia.org/wiki/Polar_coordinate_system

[51] Wolfram mathworld: Cylinder.
url: https://mathworld.wolfram.com/Cylinder.html

[52] Wolfram mathworld: Sphere.
url: https://mathworld.wolfram.com/Sphere.html

[53] Wolfram mathworld: Coordinate System.
url: https://mathworld.wolfram.com/CoordinateSystem.html

[54] Wolfram mathworld: Tensors.
url: https://www.wolfram.com/broadcast/video.php?c=311&v=107

[55] Wolfram mathworld: Sphere.
url: https://mathworld.wolfram.com/Tensor.html

[56] Wolfram.alpha mathworld: Sphere.
url: https://www.wolframalpha.com/input/?i=tensor

[57] Wolfram mathworld: MetricTensor.
url: https://resources.wolframcloud.com/FunctionRepository/resources/MetricTensor/