Tensors in APL
A Notebook

Introduction

Somewhere along the way, I heard Ken Iverson, in a discussion about APL, respond to the question: "*So, what about tensors?*" He had no hesitation. I think he said "*What do you mean? It's all there*." I sensed that he meant what he was saying. Needless to say, no one challenged him.

I studied tensor analysis at university and certainly enjoyed the topic, largely for the slick way the notation worked. But, of course, an undergraduate course is carefully crafted to steer clear of the more difficult bits.

A little later, I learned APL and had the same reaction: what a wonderful notation. Over the years, I've used APL to do a lot of satisfying work. But, always lingering there was the topic of tensors. I never saw anyone follow up on Ken's comment, so I decided, in my retirement, to take it on. This has not been straightforward. I've had at least a couple of false starts on this in the past ten years. But now I believe I have something that's presentable. I only wish I could get Ken's reaction.

In his paper from 1979 "*The Derivative Operator*", Ken referred to a text by S. Sokolnikoff "*Tensor Analysis, Theory and Applications*". This is a truly remarkable book for an APL enthusiast. It contains many of the ideas that form APL's foundation. I've drawn on this for a good deal of the material that follows.

But, I would be remiss if I did not also acknowledge Paul Dirac's "*General Theory of Relativity*". Dirac had an amazing talent to make the complex simple. His book is just 69 pages.

So, this is just an introduction to a large and important topic. I had to pick a point to stop and I chose the covariant derivative. Along the way, I tried to be rigorous, but I'm not a mathematician, so I'm open to challenge.

Mike Powell
mdpowell@gmail.com
July 2024

The APL environment

All of the text in the APL385 Unicode font is executable in APL. The particular APL used here is Dyalog APL 18.2 with:

```
⎕io←0
⎕pp←6
]boxing on
```

Dyalog APL is freely available for non-commercial use at www.dyalog.com.

Rank

Nowadays, the rank of a monadic function is defined as a single number, the *argument rank*. This is the rank of the array of greatest rank that does not produce a frame. However Iverson chose to prefix this value with the rank of the result so produced. In what follows, this will be helpful, so we will adopt Iverson's two integer definition of function rank: we will assume that a reference to "rank `m n`" refers to an argument rank of `n` producing a result rank of `m`. In what follows, we are almost always dealing with arguments that are vectors. So if we refer to "rank `m`", we really mean "rank `m,1`".

Some useful definitions

Utility functions

First, we have functions that don't do a lot more than give familiar names to APL functions:

`min←{⌊/,ω}`	*Minimum*
`max←{⌊/,ω}`	*Maximum*
`num←{×/ρω}`	*Number*
`sum←{+/,ω}`	*Sum*
`mean←{(sum÷num)ω}`	*Mean*
`sop←{+/,α×ω}`	*Sum of product*
`ssq←{ω+.×ω}`	*Sum of squares*
`rnd←{α×⌊0.5+ω÷α}`	*Round*
`disp←{⊂⍤2⊢ω}`	*Display higher rank array*
`hyp←{0.5*⍨ssq ω}`	*Hypotenuse*
`sin←{1○ω}`	*Sine*
`cos←{2○ω}`	*Cosine*
`tan←{3○ω}`	*Tangent*
`atan←{¯3○ω}`	*Arc tangent*
`id←{(ιω)∘.=ιω}`	*Identity function*
`lm←{(ιρω)∘.≥ιρω}`	*Lower mid array*
`alt←{ω×(ρω)ρ1 ¯1}`	*Alternate*
`sh←{(α⌽ιρρω)⍉ω}`	*Shift axes*
`xp←{α×⍤(-(ρρα)⌊ρρω)⊢ω}`	*Extended product*

As derivatives play such a large part in what follows, it's useful to include the derivatives of two of the trigonometric functions:

`dhyp←{ω÷hyp ω}`	*Derivative of* `hyp`
`datan←{÷1+ω×ω}`	*Derivative of* `atan`

Then we have two operators, defined by Iverson in his paper on derivatives. These will see a lot of action:

`ip←{(αα ω)+.×ωω ω}`	*Inner product operator*
`op←{(αα ω)∘.×ωω ω}`	*Outer product operator*

Comparing arrays

In order to check our work in the examples, we will need to deal with comparing arrays of numbers which are almost the same. We'll define a function `comp` that returns a scalar measure of how close two arrays are.

```
comp←{
  (ρρα)≢ρρω:0
  (ρα)≢ρω:0
  a b←,¨α ω
  mean 0=0.99 1.01⍸a÷b+2×a×b=0}
```

This works well most of the time. However, it does not do so well if our arrays have elements which are both `0` or that should be zero but instead are just very small (due to numerical error). In those cases, `comp` will have to be replaced with something like `comp⍤{0.000001⌈ω}` or `comp⍤{0.01 rnd 1000000×ω}`.

Spaces & Fields

The Windy website

Here's an image from the windy.com website, showing what's happening with the wind in early 2023 at the surface in the Pacific Northwest. Most of the wind activity is over the ocean with just light breezes inland.

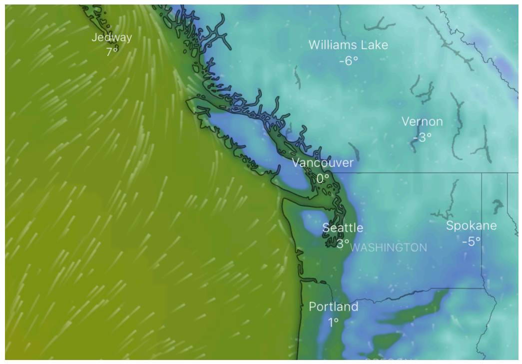


Figure 1, the windy.com website

As wind velocity is a vector, this map shows both the wind's direction and magnitude as arrows. Obviously if this was done at every observation point the arrows would overlap and obscure each other. So the website rather craftily shows samplings of the data. At any one moment, only a small amount of the available data is displayed. These arrows then fade away to be replaced by a new sample. Doing it this way produces a nice impression of wind movement. The background colouring just shows the surface temperature in degrees Celsius.

So this map is actually showing two values at each point, one of which is a vector and the other a scalar. These are both fields.

windy.com provides two other controls which allow the observer to delve into a full four dimensional space. One controls altitude and the other time. The altitude slider lets us move up through the atmosphere and look at the wind field all the way from the surface up to 13.5 km. The time slider lets us look ahead up to 10 days in advance:

Taken together these maps give sailors an idea of wind strength and direction today and tomorrow. As boats sail basically in two dimensions (plus time), the altitude component of wind is not so important. That's impressive. Let's analyze what's going on a bit more closely.

The space we're presented with appears to be rectangular with four dimensions x, y, z and t. The two dimensional map we can look at uses the longitude and latitude as the x and y axes. The z axis corresponds to altitude and t is the time axis. Note however that, as the Earth is approximately a sphere, our x and y coordinates are really the projections of latitude and longitude values onto a plane.

Spaces and Fields

In a space of N dimensions, a *point* is represented as a vector of length N. For example, a random point could be produced with:

```
point←?N⍴1000
```

If we are interested in a *collection* of M random points, we can get them with:

```
collection←?(M,N)⍴1000
```

We can create a collection of points which are continuously connected with a generating function. This is known as a *curve*. Such a curve has an infinite number of points and is produced by applying the generating function to successive values of a parameter u as it varies through a range of values. Each application of the function is to a scalar and it produces a single point in N-space. For example, in two dimensions we can create a curve with the function {⍵*1 2}. We are unable to deal with infinities in APL, but we can show some of the points on the curve:

```
      start←1 ⋄ mid←5 ⋄ end←9
      curve←{⍵*1 2}
      curve⍤0⊢start,mid,end
1 1
5 25
9 81
```

In spaces of higher dimension, we can generate other collections of points. We do this by using generating functions which take vectors of several u parameters as their arguments. These collections of points are known as *subspaces* of the N dimensional space. In a space of N dimensions, we can produce subspaces of dimension 1 to N-1. In particular, if the dimension of the subspace is N-1, the collection of points is known as a *hypersurface* of the N dimensional space.

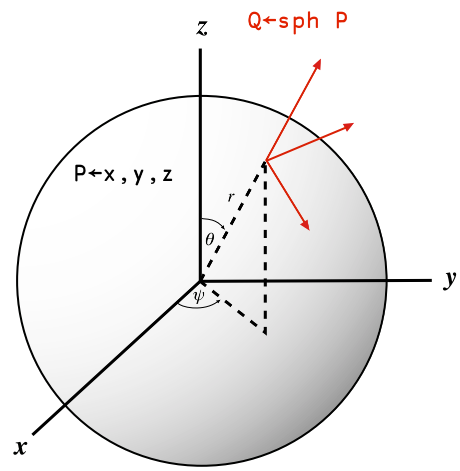
In anticipation of discussions regarding relativity, we'll sometimes use the term *frame* to mean a space of four dimensions with components *x, y, z* and *t*.

Making a measurement

When an observer takes a measurement, this happens by applying a (usually monadic) function to the coordinates of a point. The result obtained is a regular APL array, maybe a scalar, vector, matrix or of higher rank. When we collect together all the results obtained by the measurement function at each point in the coordinate space, that's a *field*. Then it makes sense to talk about a *scalar field*, a *vector field* etc.

The Sphere

A good example of a space and a subspace is to be found with a sphere. Here's one of `3` dimensions:



The coordinates x, y and z define a `3` dimensional space of rectangular coordinates. We assume that x, y and z are real numbers and extend continuously throughout some domain, which might be infinite.

A sphere is a subspace of that with coordinates r, θ and ψ. As a sphere is a solid, the test for whether points are part of that solid is given by $r^2 \geq x^2 + y^2 + z^2$. Our APL definition of this is:

```
    sphere←{α≥ö ssq ω}                    Contents of a sphere

    13 sphere 3 4 5                       Is the point 3 4 5 within a sphere of radius 13?
1
```

The surface of a sphere is another subspace of the Cartesian space. The points that make up this collection are defined by $r^2 = x^2 + y^2 + z^2$ with a function `surface`:

```
    surface←{α=ö ssq ω}                   Surface of a sphere

    13 surface 3 4 5                      Is the point 3 4 5 on the surface?
0
    13 surface 3 4 12                     3 4 12 is on the surface.
1
```

On the surface of a sphere, we can identify a local frame of coordinates at any point. These are shown in red in the diagram and represent the directions of increasing r, θ and ψ. On the sphere, the r axis is normal to the surface and the θ and ψ axes are tangential to the surface.

Spherical coordinates

We can transform back and forth between the spatial components of rectangular and spherical coordinates with:

```
   sph←{x y z←ω                                    Rectangular to spherical
     (hyp ω),atan((hyp x,y)÷z),y÷x}
```

And we can go in the reverse direction with:

```
   sphi←{r theta psi←ω                             Spherical to Rectangular
     r×((sin theta)×(cos,sin)psi),cos theta}
```

For example:

```
   ⍉xyz←sphi 3.60746 1.52521 3.10805
5 0.9 0.4
   sph xyz
3.60746 1.52521 3.10805
```

Cylindical coordinates

Perhaps, if we're studying the mechanics of a fluid in a pipe, we might choose cylindrical coordinates.

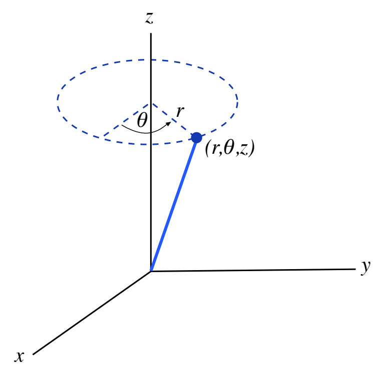


Figure 3, Cylindrical cooordinates

And we can switch back and forth with:

```
   cyl←{x y z←ω                                    Rectangular to cylindrical
     (hyp x,y),(atan y÷x),z}

   cyli←{r theta z←ω                               Rectangular to Cartesian
     (r×(cos,sin)theta),z}
```

The twisted space

Just in case it's needed below, here's one more space formed by a transformation function whose metric is not diagonal.

```
   tw←{(+\ω)*2}                                    Twisted
   twi←{t←ω*0.5 ⋄ t-¯1↓0,t}                        Its inverse
```

Analytic derivatives

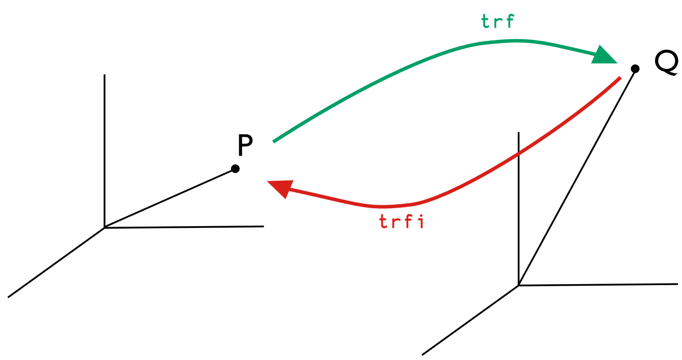
It will be useful later on to have analytic expressions for the derivatives of `sph`, `sphi`, `cyl` , `cyli`, `tw` and `twi`. These are:

```
dsph←{x y z←ω ⋄ z←÷z ⋄ h←hyp x,y                    Derivatives of sph and sphi
  r←dhyp ω
  r,←(z×(datan z×h)×(dhyp x,y),-z×h)
  3 3⍴r,(datan y÷x)×(-y÷x×x),(÷x),0}
dsphi←{r theta psi←ω
  a b←(sin,cos)theta ⋄ c d←(sin,cos)psi
  3 3⍴(a×d),(r×b×d),(-r×a×c),(a×c),(r×b×c),(r×a×d),b,(-r×a),0}

dcyl←{x y z←ω ⋄ t←y÷x                                Derivatives of cyl and cyli
  3 3⍴(3↑dhyp 2↑ω),(3↑(datan t)×((-t),1)÷x),0 0 1}
dcyli←{r theta z←ω ⋄ a b←(cos,sin)theta
  3 3⍴a,(-r×b),0,b,(r×a),0 0 0 1}

dtw←{2×(⍳m ω)×⍉(2⍴⍴ω)⍴+\ω}                          Derivatives of tw and twi.
dtwi←{t←(2⍴⍴ω)⍴0 ⋄ (0 0⍉t)←0.5×ω*-0.5 ⋄ t-¯1 0↓0⍪t}
```

Conventions



To simplify matters, we'll adopt some conventions. We'll assume that:

`P` is a point in a space represented by a vector of rectangular coordinates. This point will be labelled `Q` in a second frame. The coordinates `Q` will be derived from `P` via a transform function `trf`. Alternatively, `P` can be derived from `Q` via the inverse `trfi`. (Of course, in order for this to work, `trf` must be invertible.)

```
    P←3 1 4 2
    trf←{ω*2} ⋄ trfi←{ω*0.5}
    Q←trf P
    P≡trfi trf P
1
```

Tensors

What is a tensor?

A tensor is a field of values produced by the application of a measurement function to each point in a coordinate space. In APL terms, the result produced by the measurement function at a point is a simple (i.e. non-boxed) array.

However, tensors are a bit more than merely a field of APL arrays. Whether the measurement function represents a tensor depends on the law of transformation of that function from one coordinate system to another. More on that shortly.

Because of lot of scientific analysis focuses on relationships at a single point, it has been customary to refer to a tensor as a data value rather than as the function which produced it. Which is unfortunate.

Why are tensors important?

Tensors, by design, transform in predictable ways. The outcome of this is that if we can write a physical law as an equation whose terms are tensors and it is found to be true in one coordinate frame, then it is guaranteed to be true in all coordinate frames.

This is very appealing to scientists. What's the point of having a physical law if you have to restate it in a different form for every different observer? Of course, the classic example of this is Einstein's work on General Relativity, which was only possible with tensors. Sokolnikoff puts it this way:

> "*Since tensor analysis deals with enitities and properties that are independent of the choice of reference frames it forms an ideal tool for the study of natural laws. Indeed, whether a logical deduction based on a conglomerate of observational facts deserves the name of a natural law is often determined by the generality of such a deduction, and by its validity in a sufficiently wide class of reference systems.*"

Einstein notation

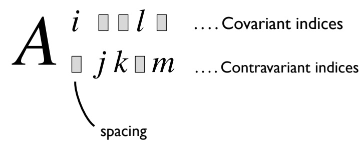
The notation for tensor objects, adopted by Einstein and many others, employs indices, both as superscripts and subscripts. The rank of an object is equal to the number of indices. So:

ϕ is a scalar
x^r , x_s are vectors
a_{rs} is a matrix
...
$m_p{}^{qr}{}_s$ is a rank 4 array

A similar notation is used for functions. For example f^r is a function that returns a vector as its result.

The use of superscripts and subscripts is most important. As will be seen shortly, the placement of an index determines how values change under coordinate transformations. A subscripted index is known as a *covariant* index; a superscripted index as a *contravariant* index. Objects that have both types of index are known as *mixed*.

The positioning of the indices is also important and spacing needs to be inserted during typesetting to correctly locate the indices. Text that contains a superscript index directly above a subscript index is ambiguous as the order of the indices is unclear. Here's an example of a rank 5 array with properly spaced indices.



Einstein notation is both declarative and functional. It is declarative because it tells us the rank of an object and the variance quality of each dimension. And it is functional because it prescribes functions that should be applied to that tensor object. There are three functions to consider:

(a) Outer product
If two tensors are written without an intervening function, an outer product is assumed. Thus $y^r z^s$ is equivalent to `y∘.×z`.

(b) Transposition
If a tensor is written with some of its indices in an order that is not ascending, this indicates that a transpose is to be applied to return the indices to their natural order. For example, A_{ijlmk} implies a transpose of the last three axes of a tensor, A_{ijklm}.

(c) Contraction
The *summation convention* means, when an index is repeated in a term, a summation with respect to that index is understood. This is known as *contraction* and applies to one superscript and one subscript. For example, $m_{rs} v^s$ is equivalent to `m+.×v`. But note that $m_{sr} v^s$ specifies a slightly different product, `v+.×⍉m`.

APL operations with tensors

As tensors are just APL arrays, together with information about how they transform, we can use all of APL's functions and operators in expressions to manipulate them. For example, p^s+q^s, $M_{ij}\times N_{ij}$ and e^{ϕ} are all valid tensor expressions in Einstein notation and they have APL counterparts `p+q`, `M×N` and `*phi`. All the arithmetic and structural functions are available. We can reshape tensors, take pieces of them, join them together or apply a reduction or scan function.

Note, however, that Einstein notation is less expressive than APL and many operations that can be expressed in APL have no counterpart in Einstein notation. For example, `0 1 1⍉M` produces the trace of the second and third axes of an array `M`, reducing the rank by `1`. About the closest you can get with Einstein notation is the contraction $M_{ij}{}^{j}$, but this does a summation of the trace and reduces the rank by `2`.

Tensor value

Let's define a function to emulate conventional index notation. We'll assume that its right argument is an array and that any outer products used in its construction have been performed. The function we define will need to handle both contractions and transposes. The left argument will be a vector of indices which specify the operations to be done in the same way as Einstein notation. Let's name the function `val`.

For example, in order to produce the contraction specified as $M_{ij}{}^{i}{}_{k}$, we would use a left argument to `val` of `'ijik'`. If we just wanted to produce a transpose of a rank `3` array specified as M_{kij}, we would use a left argument of `'kij'`. And, performing two contractions together with a transpose could be done with `'ijlmkji'`.

Here's a definition for `val`:

```
paired←{⍵{⍺∊(2=+/⍵∘.=⍺)/⍵}∪⍵}                  Paired indices?

dense←{(({⍵[⍋⍵]}∪⍵)⍳⍵}                          Dense integers

val←{                                            Tensor Value
  b←paired ⍺ ⋄ c←~b                              Find the indices that represent contractions.
  x←⍳⍴⍺ ⋄ s←b/⍺
  x[(⍸c),⍸b]←(dense c/⍺),(+/c)+(∪s)⍳s            Calculate the indices to be used in the result.
  +/,⍤(0.5×⍴s)⊢x⍉⍵}                              Transpose the argument and sum the final axes.
```

`val` takes a left argument of a vector of indices and performs the contractions implied in the indices; along the way it also performs any transposes specified by the non-contracting indices.

Note that the indices we choose do not have to form a sequence or they can be Greek letters or even numbers:

```
   a←?4 3 2 5 3 4⍴100
   t←'ijklji'val a
   t≡'acdgca'val a
1
   t≡'αερωεα'val a
1
   t≡4 7 2 5 7 4 val a
1
```

The indices for `val`

The left argument to `val` specifies both contractions and transpositions.

The values we choose for contraction indices are unimportant. They are destined to disappear and we can spot them as they appear in pairs in the left argument. For our purposes, any values will do.

However, for indices to represent transformations, we need to know their "natural" order. For a numeric argument, this is just ordinary arithmetical order. So `3 2` represents a transposition but `2 3` does not. If the index vector is character, we'll rely on `⎕ucs` to provide the ordering – and that's exactly what happens in the function `dense`.

`dense` works on both numeric and character arguments, returning values drawn from consecutive integers starting at `0`. For example:

```
   dense 3 0 6 3 1
2 0 3 2 1
   dense 2 1.7 4.2
1 0 2
   dense'ijjkqli'
0 1 1 2 4 3 0
```

Although we will not make use of this, `dense` works as expected on more general APL values:

```
   dense'first' 'third' 'second' 'ultimate'
0 2 1 3
```

This is all well and good but what if we'd like to work with a character index vector rater than a numeric one? First we'll have to decide on which characters to use. Here's a suggestion:

```
latin←'abcdefghijklmnopqrstuvwxyz'
greek←'αβγδεζηθικλμνξοπρστυφχψω'                 Currently unused
dummy←'.-*∘=+∨∧'
```

```
   cix←{                                          Character index vector
     a←paired ⍵ ⋄ b←~a                            starting at 'i'
     ((8⌽latin),dummy)[(a\26+dense a/⍵)+b\dense b/⍵]}

   cix'qaikqb'
.ikl.j
```

Note that cix is limited to `26` transposition indices and `8` contraction pairs. More than enough for our purposes.

Transpositions

Note that the transpositions specified in Einstein notation work in the same way as APL's dyadic transform.

APL's dyadic transform uses its left argument to specify where each axis in the right argument should be placed in the result. So, for example, if the second element of the left argument is `4`, this means that the second axis of the right argument will be moved to position `4` in the result. `val` works in exactly the same way. Here's an example.

```
   a←2 3 4 5 6⍴7
   ⍴t←0 4 1 2 3⍉a
2 4 5 6 3
   t≡0 4 1 2 3 val a
1
```

What if we have a dyadic transpose of an array, specified with `⍉`, and we'd like to convert this to use `val`? If we start with `k⍉a`, the equivalent is just `k val a` – but with the caveat that `k` must not specify a trace as that cannot be represented in Einstein notation.

Contractions and dummy indices

Indices that are repeated in the left argument specify contractions. They must appear in pairs and, after the contractions are made, the corresponding axes disappear. Here's an example that demonstrates the reduction of a rank `6` array to a matrix by applying two contractions. In the result, four axes are removed and we are left with a `2` by `5` matrix.

```
   ⎕rl←16807
   a←?4 3 2 5 3 4⍴100
   ⊣t←'ijklji'val a
721 762 785 598 686
418 614 762 492 627
```

Where the left argument to `val` implies a contraction, we are free to choose any character we want, as long as it's not used elsewhere. That's known as a dummy index. For example:

```
   t≡'.-ij-.'val a
1
```

Successive applications of `val`

What if we use `val` twice on an array? As we're just doing contractions and transpositions, we ought to be able to simplify and use `val` just once. What we'd like to do is replace `x1 val x0 val a` with just `x val a`.

First, an observation. The length of `x1` cannot be greater than than the length of `x0` (as `val` can never increase the rank of its array argument).

Let's work through an example:

```
   a←?3 2 4 2 5⍴9
   ⍴t←'kij'val'i.k.j'val a
5 4 3
```

The first `val` to be executed does a contraction and rearranges the axes so that the result is of shape `3 5 4`. This comes about because, after the removal of the contraction axes, the shape is `3 4 5` and the indices `'ikj'` exchange the last two axes. The second use of `val` causes a further rearrangement of these three axes giving a result of shape `5 4 3`.

We can simplify this by observing that the application of `'kij'val` causes the first axis to be moved to the end. That means we can achieve the same result with one application of `val` with an argument of `'k.j.i'`.

```
   t comp'j.i.k'val a
1
```

Merging index vectors

Notice that it was not too difficult to combine the `'kij'` and `'ikj'`arguments to `val` in the expression above. That's because there were no contractions to deal with. Where there are contractions, we have to be a little more careful. Here's a function `merge` that will combine two `val` index arguments into one.

```
   merge←{
     x0←dense ⍵ ⋄ b←~a←paired x0                   Merge two index vectors
     x1←dense ⍺ ⋄ d←~c←paired x1
     t←a\26+dense a/x0
     s←x1
     s[⍸c]←26+(0.5×+/a)+dense c/x1
     s[⍸d]←dense d/x1
     t+b\s[dense b/x0]}
```

For example:

```
   a←?3 4 5 6 3 2 4 1⍴9
   x0←'.inl.kmj'
   x1←'.ilk.j'
   ⍴b←x1 val x0 val a
1 5 6 2
   ⊣x←x1 merge x0
26 27 1 2 26 3 27 0
   cix ix
.-jk.l-i
   b≡x val a
1
```

Derivative of `val`

As the use of `val` involves nothing more than the rearrangement or summation of values provided in the right argument, we should expect that its derivative behaves much like the derivative of a sum. Here's an example involving a contraction:

```
   {'..'val ⍵∘.×⍵}∆ 2 7 5
4 14 10
   '..i'val{⍵∘.×⍵}∆ 2 7 5
4 14 10
```

And here's one involving a transposition:

```
   ρa←{'ji'val ω∘.×ω}∆ 2 7 5
3 3 3
   a comp'jik'val{ω∘.×ω}∆ 2 7 5
1
```

Associativity of `val`

It is clear that `val` is associative under addition. So,

```
   x val a+b ←→ (x val a)+x val b  ......................................... [1]
```

Inner and outer product

Naturally, APL's inner product `+.×` can be expressed using `val`. Here are some common cases:

m	n	m+.×n
Vector	Vector	'..'val m∘.×n
Matrix	Vector	'i..'val m∘.×n
Matrix	Matrix	'i..k'val m∘.×n
Rank 3 array	Vector	'ij..'val m∘.×n

Useful relationships between `trf` and `trfi`

As shorthand, we'll define:

```
   T←trf ∆ ⋄ t←T P
   TI←trfi ∆  ⋄ ti←TI Q
```

There is a relationship between `T` and `TI` which will be useful later on. Consider the expression `trfi∘trf P`. If we take its derivative the result will be a unit matrix:

```
   trfi∘trf ∆ P
»  (trfi ∆ trf P)+.×trf ∆ P                    Derivative of a composition

   (TI Q)+.×T P  ←→  idρP  ................................................ [2]
```

Similarly, if we start with `trf∘trfi ∆ q`, we get:

```
   (T P)+.×TI Q  ←→  idρP  ................................................ [3]
```

We can now go one step further and examine the relationship between the derivatives of `T` and `TI`. Consider the expression `(T∘trfi)ip TI ∆ Q`. This is just the derivative of `idρP` which is of the same shape, but all zero. We can expand this as follows:

```
   (T∘trfi)ip TI ∆ Q
»  ((T∘trfi Q)+.×TI ∆ Q)+¯1 sh(1 sh(T∘trfi)∆ Q)+.×TI Q
»  (t+.×TI ∆ Q)+¯1 sh(1 sh(T∘trfi)∆ Q)+.×ti
»  (t+.×TI ∆ Q)+¯1 sh(1 sh(T ∆ trfi Q)+.×trfi ∆ q)+.×ti
»  (t+.×TI ∆ Q)+¯1 sh(1 sh(T ∆ P)+.×ti)+.×ti
»  (t+.×TI ∆ q)+¯1 sh('jk..i'val(T ∆ p)∘.×ti)+.×ti
»  (t+.×TI ∆ Q)+¯1 sh'i-..k-j'val(T ∆ P)∘.×ti∘.×ti
»  (t+.×TI ∆ Q)+'i-..k-j'val(T ∆ P)∘.×ti∘.×ti
```

As we started with an expression producing a zero array, we can write:

```
   (t+.×TI ∆ Q)  ↔  -'i-..k-j'val(T ∆ P)∘.×ti∘.×ti
```

If we form an inner product on the left with `ti`, we now have:

```
   TI ∆ Q  ↔  -'*-.i*.k-j'val(T ∆ P)∘.×ti∘.×ti∘.×ti  ............................. [4]
```

And, had we started with `(TI∘trf)ip T ∆ P`, we'd get:

```
   T ∆ P  ↔  -'*-.i*.k-j'val(TI ∆ Q)∘.×t∘.×t∘.×t  ................................ [5]
```

Exercise

Synge and Schild [3] include an exercise (at page 8) which demonstrates working with tensors: If $\phi = a_{rs}x^r x^s$, show that $\partial\phi/\partial x^r = (a_{rs} + a_{sr})x^s$. (This assumes that a_{rs} is a constant.)

First we'll set up an example just to show that the APL calculations work out:

```
   a←4 4⍴3 1 4 2 9 5 0 6 7 3 1 4 2 9 5 0
   x←6 2 4 3
   phi←{'.-.-'val a∘.×⍵∘.×⍵}
   phi x
822
   phi ∆ x
112 137 107 90
   'i..' val(a+⍉a)∘.×x
112 137 107 90
   (a+⍉a)+.×x
112 137 107 90
```

Now let's use some of the derivative rules from Appendix B and the relationships we've set out for tensor evaluations to prove the expression for `phi ∆ x`:

```
   phi ∆ x
»  {'.-.-'val a∘.×⍵∘.×⍵}∆ x
»  '.-.-i'val{a∘.×⍵∘.×⍵}∆ x
```

The derivative of a contraction is the contraction of the derivative.

The derivative term `{a∘.×⍵∘.×⍵}∆` is of a function incorporating two outer products. Expanding this as the derivative of an outer product with the constant `a` yields two terms, one of which has the derivative of the constant `a`. As this term is zero, we are left with:

```
»  '.-.-i'val a∘.×{⍵∘.×⍵}∆ x
```

We can expand the `{⍵∘.×⍵}∆` term as the derivative of an outer product. It is helpful to define a unit matrix with `unit←idρx`.

```
»  '.-.-i'val a∘.×(x∘.×unit)+0 2 1⍉unit∘.×x
»  '.-.-i'val (a∘.×x∘.×unit)+a∘.×0 2 1⍉unit∘.×x
```

We can now introduce a transpose to make the right term refer to the product `(⍉a)∘.×x∘.×unit`:

```
»  '.-.-i'val(a∘.×x∘.×unit)+1 0 3 2 4⍉(⍉a)∘.×x∘.×unit
»  ('.-.-i'val a∘.×x∘.×unit)+'.-.-i'val 1 0 3 2 4⍉(⍉a)∘.×x∘.×unit
```

Now we can combine the `1 0 3 2 4⍉` in the right term by combining it with the index argument to `val`. This gives:

```
»  ('.-.-i'val a∘.×x∘.×unit) + '-.-.i'val(⍉a)∘.×unit∘.×x
```

The dummy indices used in the second term can be exchanged giving:

```
»  ('.-.-i'val a∘.×x∘.×unit) + '.-.-i'val(⍉a)∘.×x∘.×unit
»  '.-.-i'val(a∘.×x∘.×unit) + (⍉a)∘.×x∘.×unit
»  '.-.-i'val(a+⍉a)∘.×x∘.×unit
```

As `a+⍉a` is guaranteed to be symmetric, we can exchange the first two indices:

```
»  '-..-i'val(a+⍉a)∘.×x∘.×unit
»  '--i'val(a+⍉a)+.×x∘.×unit               Definition of inner product
»  '--i'val((a+⍉a)+.×x)∘.×unit             Inner product is distributive
»  (a+⍉a)+.×x                              Definition of inner product
```

Variance

Invariance

Some mathematical objects require no adjustment under a coordinate transformation. Sokolnikoff puts it this way:

> "*An object, whatever its nature, is an invariant, provided that it is not altered by a transformation of coordinates*."

But surely this cannot be right? Consider the simple case where a point with coordinates `P` in a space measured in metres is viewed from a space which makes measurements in centimetres. In that second space, we're going to see coordinates of `100×P`, which are certainly different values. The discrepancy is explained by noting that although the values we see in the second space are numerically different, the point itself is unchanged. This means that:

- The points themselves do not change. It makes no difference whether a point is expressed in the coordinates of the first space or of the second. Both references are to the same point.
- Vectors are determined by the difference between a pair of points. Again, it does not matter whether we use the coordinates of the first space or of the second. The vector remains unchanged.
- A set of points, such as those forming a curve or a surface, is also invariant.

In general if we are taking a measurement in the P space with a function `f`, then the appropriate function to use in Q is `f∘trfi`.

Contravariance

The differential element

Fairly frequently, analysis involves taking a tiny (in the limit, infinitesimal) step away from a point. If we're just talking about a single point, we can simply add a vector of small values to those of the point to effect the translation. For example, at the point `P←123 456 789`, we could use an increment of `dP←0.1 0.1 0.1` to make a small step to the point `123.1 456.1 789.1`. But, if instead of a single point we have an entire field to deal with, we need to be more flexible. After all one of those points might be `1 1 1` and now our suggested value for `dP` is no longer small. A better way to proceed is to use a function `diff` to be applied to the coordinates of a point which will give us a suitable small value for the differential element.

In the general case, choosing the function `diff` can get complicated. But for now, we'll put any concerns to one side and simply define `diff` as:

```
    diff←{0.000001×ω}
```

Transformation of the differential element

Suppose we have two points `P` and `P+dP`. We can transform these points with a function `trf` to points `Q` and `Q+dQ`. Having done so, what is the relationship between `dQ` and `dP`?

As `dQ` is just the difference between the transformed points, we have:

```
    dQ
»   (Q+dQ)-Q
»   (trf P+dp)-trf P
```

As dP is small relative to P, we can expand trf about P using a first order Taylor expansion. This produces

```
»  ((trf P)+(trf ∆ P)+.×dP)-trf P
»  (trf ∆ P)+.×dP
»  (T P)+.×dP
```

Here's an example:

```
   trf←{⍵*1.2} ⋄ trfi←{⍵*÷1.2} ⋄ T←trf ∆ ⋄ TI←trfi ∆
   P←3 1 4 2 ⋄ q←trf P
   ⊣dP←diff P
0.000003 0.000001 0.000004 0.000002
   ⊣dq←(trf P+dP)-trf P
0.00000448463 0.0000012 0.00000633364 0.00000275688
   (T P)+.×dP
0.00000448463 0.0000012 0.00000633364 0.00000275688
   T iP diff P
0.00000448463 0.0000012 0.00000633364 0.00000275688
```

This last expression shows how diff must be varied for the Q space. But it does so using P as its argument. What if we'd rather see Q as the argument? This is just:

```
   T ip diff∘trfi Q
0.00000448463 0.0000012 0.00000633364 0.00000275688
```

The transformation of the differential element is the prototype for the definition of the contravariant tensor.

Definition

A rank 1 1 function f is said to produce a *contravariant vector field*, if it transforms to be:

```
   'i..'val T op f∘trfi  ........................................................  [6]
```

when applied to coordinates in Q. This is $\partial x'^r/\partial x^s\ V^s$ in Einstein notation.

Taking two steps

The example above demonstrates how a tensor field produced by a function diff in P space transforms to Q space. What if we added on a second transformation that takes us back to P space from Q space? That just requires us to use trf and TI in place of trfi and T for the second transformation.

```
   'i..'val TI op('i..'val T op diff∘trfi)∘trf p
0.000003 0.000001 0.000004 0.000002
```

Covariance

Transformation of the gradient of a function

Consider a function phi which acts on a vector of coordinates P. For example:

```
   phi←{⍵+.*0.8 0.9 1 1.1}
   phi P←3 1 4 2
9.55177
```

We can compute the gradient for phi with:

```
   phi ∆ P
0.642193 0.9 1 1.17895
```

If we now do the equivalent operation in the Q space, how is the gradient calculated there related to the gradient in P? We can answer this question by observing that the function `phi` needs to be modified to `phi∘trfi` before use in the second space. Doing so, we have:

```
   trf←{⍵×1.4}
   trfi←{⍵×÷1.4}
   Q←trf P
   phi∘trfi ∆ Q
»  (phi ∆ trfi Q)+.×trfi ∆ Q                    The derivative of a composition
»  (phi ∆ P)+.×TI Q
```

This shows that the gradient calculated in the second space can be obtained from the gradient in the first space by taking an inner product with the matrix `TI Q`.

The gradient of a function serves as the prototype for a covariant tensor field and, in general, we can use a different function f in its place. Doing so, we have:

```
   (f P)+.×TI Q
»  (f∘trfi)ip TI Q
»  '.i.'val TI op(f∘trfi)Q
```

Definition

A rank `1 1` function `f` is said to produce a *covariant vector field*, if it transforms to be:

```
   '.i.'val TI op(f∘trfi)  ..................................................  [7]
```

when applied to coordinates in Q. This is $\partial x^i/\partial x'^j\ V_i$ in Einstein notation.

Higher rank tensors

We should anticipate that tensors may be of higher rank. Dirac (p. 2) describes how these transform. He says:

> *"From the two contravariant vectors A^μ and B^μ we may form the sixteen quantities $A^\mu B^\nu$ ... sometimes called the outer product ... we can add together several tensors constructed in this way to get a general tensor of the second rank, say*
>
> $T^{\mu\nu} = A^\mu B^\nu + A'^\mu B'^\nu + A''^\mu B''^\nu + \ldots$
>
> *The important thing about the general tensor is that under a transformation of coordinates its components transform in the same way as the quantities $A^\mu B^\nu$."*

So, let's follow Dirac and examine the transformation of the outer product U^iV^j. In Einstein notation, the transformed product is:

$$U'^kV'^l \quad = \quad (\partial x'^k/\partial x^i\ U^i)\ \partial x'^l/\partial x^j\ V^j$$

For contravariant vector functions `u` and `v`, this becomes:

```
   ('i..'val T op u∘trfi Q)∘.×'j--'val T op v∘trfi Q
»  'i..j--'val(T op u∘trfi)op(T op v∘trfi)Q
»  'i..j--'val T op u op T op v∘trfi Q
»  'i.j-.-'val T op T op u op v∘trfi Q
»  'i.j-.-'val T op T op(u op v)∘trfi Q
```

We can form a covariant matrix field in much the same way. In Einstein notation this is:

$$U'_kV'_l \quad = \quad (\partial x^i/\partial x'^k\ U_i)\ \partial x^j/\partial x'^l\ V_j$$

As u and v are covariant vector functions, their outer product transforms to a point Q in the second frame as:

```
   ('.i.'val TI op(u∘trfi)Q)∘.×'-j-'val TI op(v∘trfi)Q
»  ('.i.'val TI op(u∘trfi))op('-j-'val TI op(v∘trfi))Q
»  '.i.-j-'val TI op(u∘trfi)op TI op(v∘trfi)Q
»  '.i-j.-'val TI op TI op(u∘trfi)op(v∘trfi)Q
»  '.i-j.-'val TI op TI op(u op v∘trfi)Q
```

Definitions

We can use these results to form definitions for the transformation of contravariant and covariant matrix fields, For f, a rank 2 1 function of the coordinates, we define

for a contravariant matrix field:

```
   'i.j-.-'val T op T op f∘trfi  .................................................  [8]
```

for a covariant matrix field:

```
   '.i-j.-'val TI op TI op(f∘trfi)  ...............................................  [9]
```

when applied to coordinates in Q.

Rank 3 and higher

We can generalize these relations to rank 3 tensors. For f, a rank 3 1 function of the coordinates, we define

for a contravariant rank 3 field:

```
   'i.j-k*.-*'val T op T op T op f∘trfi  .........................................  [10]
```

for a covariant rank 3 field:

```
   '.i-j*k.-*'val TI op TI op TI op(f∘trfi)  .....................................  [11]
```

when applied to coordinates in Q.

And, as an example, here are two ways of writing the transformation of the rank 4 contravariant tensor z formed as the sum of terms such as $U^\alpha V^\beta W^\gamma X^\delta$:

```
   'i.j-k*l∘.-*∘'i.jk.'val'val T op T op T op T op z∘trfi

   0 4 1 5 2 6 3 7 4 5 6 7 val T op T op T op T op z∘trfi
```

Mixed tensors

It's no surprise that a function producing a field can transform with a mixture of covariant and contravariant parts. For example, suppose we have vector fields produced by functions u and v. Assume that u forms a contravariant vector field and v forms a covariant one. Then their individual transformations look like this:

`'i..'val T op u∘trfi Q`	*Transformation of the contravariant vector field produced by* u
`'-j-'val TI op(v∘trfi)Q`	*Transformation of the covariant vector field produced by* v

Now consider the outer product of u and v. How does this transform?

We have:

```
   ('i..'val T op u∘trfi Q)∘.×'-j-'val TI op(v∘trfi)Q
»  'i..-j-'val(T∘trfi)op(u∘trfi)op TI op(v∘trfi)Q
»  'i.-j.-'val(T∘trfi)op TI op(u op v∘trfi)Q
```

Here's how the rank `5` mixed tensor $A^{i}{}_{jkl}{}^{m}$ transforms:

```
   'i.m+-j*k=l.-*=+'val(T op T∘trfi)op TI op TI op TI op(A∘trfi)q
```

Alternative forms

So far, we have tried to follow the way most authors write transformed tensors. For example, the transformation rule for a covariant vector field is written conventionally as $\partial x^i/\partial x'^j\, V_i$. This places the partial derivatives first, in front of the vector term. In APL, this is `'iji'val TI op(f∘trfi)q`. However, in Einstein notation, it is equally valid to write $V_i\ \partial x^i/\partial x'^j$ and then the APL equivalent is `'iij'val(f∘trfi)op TI Q`. So, there are alternatives.

It's possible to produce alternatives that make use of the inner product operator `ip`. The expressions for the vector and matrix cases are certainly of interest. However, for the higher rank cases, the expressions become more awkward.

For reference, here's a table of some equivalent forms:

Rank	Einstein Notation	APL
Scalar	S	`s`

Contravariant Vector	$\partial x'^r/\partial x^s\ V^s$	`'i..'val T op v∘trfi`
		`'.i.'val v op T∘trfi`
		`T ip v∘trfi`

Covariant Vector	$\partial x^i/\partial x'^j\, V_i$	`'.i.'val TI op(v∘trfi)`
		`'..i'val(v∘trfi)op TI`
		`(v∘trfi)ip TI`

Contravariant Matrix	$\partial x'^k/\partial x^i\ \partial x'^l/\partial x^j\ M^{ij}$	`'i.j-.-'val T op T op m∘trfi`
		`'.-i.j-'val m op T op T∘trfi`
		`T ip m ip(⍉∘T)∘trfi`

Covariant Matrix	$\partial x^i/\partial x'^k\ \partial x^j/\partial x'^l\ M_{ij}$	`.i-j.-'val TI op TI op(m∘trfi)`
		`'.-.i-j'val(m∘trfi)op TI op TI`
		`⍉∘TI ip(m∘trfi)ip TI`

Contravariant Rank 3	$\partial x^i/\partial x'^l\ \partial x^j/\partial x'^m\ \partial x^k/\partial x'^n\ T_{ijk}$	`i.j-k*.-*'val T op T op T op t∘trfi` `'.-*i.j-k*'val t∘trfi op T op T op T`
Covariant Rank 3	$\partial x^i/\partial x'^l\ \partial x^j/\partial x'^m\ \partial x^k/\partial x'^n\ T_{ijk}$	`.i-j*k.-*'val TI op TI op TI op(t∘trfi)` `'.-*.i-j*k'val(t∘trfi)op TI op TI op TI`

Associativity of tensor transformations

Contravariant transformations

Iverson includes a very useful proof dealing with two successive applications of the contravariant transformation operator. He says (at page 350):

> *"We will now illustrate the use of the operators in proofs considering the "contravariant-transform" operator CT defined, for any function F and invertible differentiable function T (both of rank 1 1), by:*
>
> *T CT F ←→ TΔ⊕F¨(T⍣¯1)*
>
> *and the proposition that CT is associative in the following sense:*
>
> *U CT(T CT F) ←→ (U¨T)CT F* "

Let's redo Iverson's proof in APL. To be consistent in our use of symbols, we'll refer to `u` and `v` in place of Iverson's `T` and `U`. This means that we'd like to establish the following:

```
(u∘v)ct f   ←→  u ct(v ct f)
```

For clarity we'll write `ui` for `u⍣¯1`, `vi` for `v⍣¯1` and define `ct` as:

```
ct←{αα Δ ip ωω∘(αα⍣¯1)ω}
```

Then,

`(u∘v)ct f`	
» `(u∘v)Δ ip f∘(u∘v)⍣¯1`	*Definition of* `ct` *for the composition* `u∘v`
» `(u∘v)Δ ip f∘(vi∘ui)`	*Inverse of the composition* `u∘v`
» `(u∘v)Δ∘(vi∘ui)ip(f∘(vi∘ui))`	*Matrix product of* `(u∘v)Δ` *with the composition of* `f` *and* `(vi∘ui)`
» `(u∘v)Δ∘(vi∘ui)ip(f∘vi∘ui)`	*Associativity of* `f∘(vi∘ui)`
» `u Δ∘v ip(v Δ)∘(vi∘ui)ip(f∘vi∘ui)`	*Derivative of the composition* `u∘v`
» `u Δ∘v∘vi∘ui ip(v Δ∘vi∘ui)ip(f∘vi∘ui)`	*Matrix product of* `u Δ∘v` *with the composition of* `v Δ` *and* `(vi∘ui)`
» `u Δ∘ui ip(v Δ∘vi∘ui)ip(f∘vi∘ui)`	*Composition of* `v` *& its inverse* `vi`
» `u Δ∘ui ip(v Δ∘vi∘ui ip(f∘vi∘ui))`	*Associative law for the matrix product* `(v Δ∘vi∘ui)ip(f∘vi∘ui)`
» `u Δ∘ui ip(v Δ∘vi ip(f∘vi)∘ui)`	*Matrix product of the composition of* `v Δ∘vi` *and* `f∘vi` *with* `ui`
» `u Δ ip (v Δ∘vi ip(f∘vi))∘ui`	*Matrix product of the composition of* `u Δ` *and* `v Δ∘vi ip(f∘vi)` *with* `ui`
» `u ct (v Δ∘vi ip (f∘vi))`	*Definition of* `u ct (...)`
» `u ct (v Δ ip f∘vi)`	*Matrix product of the composition of* `v Δ` *and* `f` *with* `vi`
» `u ct (v ct f)`	*Definition of* `u ct f`

For example:

```
   P←3 1 4 2
   f←{ω×2+ιρω}
   u←*∘3 ⋄ ui←*∘(÷3)
   v←*∘1.2 ⋄ vi←*∘(÷1.2)
   (u∘v)ct f P
21.6 10.8 57.6001 36
   u ct(v ct f)P
21.6 10.8 57.6001 36
```

Covariant transformations

As we did above for the contravariant transform operator, we can show that the successive use of the covariant transform operator with two transformations is equivalent to a single application of the operator using the composition of the two transformations.

Suppose we have a function `f` creating a covariant vector field and two transformations `u` and `v` to be applied to the coordinates, then if we define,

```
   ui←u⍣¯1 ⋄ vi←v⍣¯1
   cv←{⍺⍺∘⍵⍵ ip (⍵⍵ ∆)⍵}
```

we need to establish that:

```
   f cv (vi∘ui)   ↔   (f cv vi)cv ui
```

Here's the proof,:

» `(f∘vi∘ui) ip (vi∘ui ∆)`	*Definition of* `cv` *for the inverse of the composition* `u∘v`
» `f∘vi∘ui ip (vi ∆∘ui ip (ui ∆))`	*Derivative of the composition* `vi∘ui`
» `f∘vi∘ui ip (vi ∆∘ui) ip (ui ∆)`	*Associative law for matrix product*
» `f∘vi ip (vi ∆)∘ui ip (ui ∆)`	*Matrix product of the composition of* `f∘vi∘ui` *and* `vi ∆∘ui`
» `(f cv vi)∘ui ip (ui ∆)`	*Definition of* `cv` *for the* `v` *transformation*
» `(f cv vi)cv ui`	*Definition of* `cv` *for the* `u` *transformation*

and an example:

```
   f←{ω*1 2 3 4}
   u←{ω*1.5} ⋄ ui←{ω÷*1.5}
   v←{ω*1.2} ⋄ vi←{ω*÷1.2}
   Q←u∘v P
   f cv(vi∘ui) Q
0.0957929 0.833333 0.12091 0.458162
   (f cv vi) cv ui Q
0.0957929 0.833333 0.12091 0.458162
```

The Metric Tensor

Distance

Suppose we have a surface defined by a function of the coordinates. How do we calculate the distance between two points `P1` and `P2` on that surface?

If the surface is planar (i.e. a linear function of the coordinates) we can get the "straight line" distance between the points with:

```
hyp P1-P2
```

The result of this calculation is a scalar. More commonly authors prefer to work with the square of this value:

```
ssq P1-P2
```

What if we want the length of a path that is not "straight" but wanders around on a planar surface? Then we'll have to do a summation or integration of smaller elements to get the result. If `dP` is the vector that separates two points that are very close together, the square of the distance element is:

```
dP+.×dP
```

$$ds^2 = dp^i \, g_{ij} \, dp^j$$

Notice that in Einstein notation, we have had to introduce a rank `2` object g_{ij} to make the contractions happen. It's clearly symmetric and in a flat space only has non-zero elements on the diagonal. For ordinary Euclidean calculations g_{ij} is just a unit matrix and we could write the APL version as `dP+.×(id N)+.×dP`.

When space is curved, g_{ij} has non-zero off diagonal elements and both dp and g_{ij} become functions of the coordinates. Because of its role in determining the metric properties of space, g_{ij} is known as the metric tensor.

Transformation of the distance

As the distance is a scalar, we expect it to be invariant under a transformation of coordinates. Let's show that this is so.

We'll start with a general quadratic form $x^i \, g_{ij} \, x^j$ where `x` is a contravariant vector field produced by a function `f` and g_{ij} is a covariant matrix field. In APL this is `f ip g ip f P`. Transformed, this becomes:

```
((T P)+.×f P)+.×('.-.i-j'val g∘trfi op TI op TI Q)+.×(T P)+.×f P
```

It's worth making a few substitutions to avoid some of the parentheses:

```
a←f P ⋄ b←g P ⋄ c←T P ⋄ d←TI Q
```

Then `f ip g ip f P` transforms to:

```
»  (c+.×a)+.×('.-.i-j'val b∘.×d∘.×d)+.×c+.×a
```

We can eliminate the use of `val` by making a suitable transpose of the `b∘.×d∘.×d` term and replacing `∘.×` with `+.×`.

```
»  (c+.×a)+.×('-..i-j'val (⍉b)∘.×d∘.×d)+.×c+.×a
»  (c+.×a)+.×('-i-j'val(⍉b)+.×d∘.×d)+.×c+.×a
»  (c+.×a)+.×((⍉(⍉b)+.×d)+.×d)+.×c+.×a
```

Then,

```
»  (c+.×a)+.×(((⍉d)+.×b)+.×d)+.×c+.×a          x+.×y ←→ ⍉(⍉y)+.×⍉x
»  (c+.×a)+.×((⍉d)+.×b)+.×d+.×c+.×a            +.× is associative
»  (c+.×a)+.×((⍉d)+.×b)+.×a                    d+.×c is a unit matrix
»  (c+.×a)+.×(⍉d)+.×b+.×a                      +.× is associative
»  ((c+.×a)+.×⍉d)+.×b+.×a                      +.× is associative
»  (⍉d+.×⍉c+.×a)+.×b+.×a                       x+.×y ←→ ⍉(⍉y)+.×⍉x
»  (⍉d+.×c+.×a)+.×b+.×a                        c+.×a is a vector
»  (⍉a)+.×b+.×a                                d+.×c is a unit matrix
»  a+.×b+.×a                                   a is a vector
```

This shows that the quadratic form $x^i g_{ij} x^j$ is invariant and transforms as a tensor.

Definition of the metric tensor

Let's suppose that we calculate *ds*, the element of arc, in a space of rectangular coordinates as the square root of the scalar product of dx^i with itself. Observing that the contravariant tensor dx^i is a function of the coordinates, we can write this in APL as:

```
   diff ip diff P
```

If we then change to a different coordinate space, via a transformation `trf`, this becomes:

```
   (T ip diff∘trfi Q)+.×T ip diff∘trfi Q
»  (T ip diff)ip T ip diff∘trfi Q
»  '.--.**'val T op diff op T op diff∘trfi Q
»  '-.-.**'val diff op T op T op diff∘trfi Q
»  '∘-..**'val diff op (⍉T)op T op diff∘trfi Q
»  diff ip(⍉T)ip T ip diff∘trfi Q
```

It is from this that the definition of the metric tensor is chosen. The central expression `(⍉T)ip T` is a covariant tensor of rank `2`. It is a function of `trf`, so we can define an operator to use as `trf metric` , appropriate for any field:

```
   metric←{t←⍺⍺ ∆ ⋄ (⍉t)ip t ⍵}  ..........................................  [12]
```

Let's start with oblique axes:

```
   obl←{0 1 2 3+2 7 1 8×⍵}                     Oblique axes
   obli←{(⍵-⍳4)÷2 7 1 8}
   ⊣Q←obl P←3 1 4 2
6 8 6 19
   obl metric P
4  0 0  0
0 49 0  0
0  0 1  0
0  0 0 64
```

Oblique axes produce a metric tensor which gives values having zero for the off-diagonal elements. Now, let's see what we get with moving axes (as in Special Relativity):

```
   vel←0.2                                        vel is a velocity
   sr←{x y z t←ω ⋄ (x-vel×t),(y-2×vel×t),z,t}     Axes moving in the x-y plane
   sri←{x y z t←ω ⋄ (x+vel×t),(y+2×vel×t),z,t}
   sr metric P
 1    0    0 ¯0.2
 0    1    0 ¯0.4
 0    0    1  0
¯0.2 ¯0.4  0  1.2
```

As the `x` and `y` axes are functionally dependent on the `t` axis, the metric tensor has off-diagonal non-zero elements.

Lastly, let's examine the values produced by the metric tensors for spherical and cylindrical coordinates:

```
   sph metric 3 4 12
0.0806695 0.0542261 0.211757
0.0542261 0.112301  0.282343
0.211757  0.282343  0.852946

   cyl metric 3 4 5
0.3856 0.4608 0
0.4608 0.6544 0
0      0      1
```

Notice that for spherical coordinates, the metric tensor can have non-zero elements in all positions.

The fundamental tensors

The metric tensor defined above is also known as the *fundamental* tensor. It is a symmetric, covariant function. There is a counterpart to this function which is contravariant and is defined as its inverse.

In the P space, the value produced by the fundamental tensor at a point `P` is:

```
   g←(⍉trf ∆ p)+.×trf ∆ P
```

We can reduce this to an identity matrix, as follows:

```
   (trfi ∆ Q)+.×(⍉trfi ∆ Q)+.×g
»  (trfi ∆ Q)+.×(⍉trfi ∆ Q)+.×(⍉trf ∆ P)+.×trf ∆ p
»  (trfi ∆ Q)+.×(⍉(T P)+.×TI Q)+.×trf ∆ P
»  (trfi ∆ Q)+.×trf ∆ P                           using (T P)+.×TI Q ↔ (idρp)
»  (TI Q)+.×T P
»  idρP                                           using (TI Q)+.×T p ↔ (idρp)
```

Then we can produce the inverse of the covariant function with an operator to use as `trfi metrici∘trf` with:

```
   metrici←{t←αα ∆ ⋄ t ip(⍉t)ω}  ..........................................  [13]
```

Here's an example using cylindrial coordinates:

```
   g←cyl metric
   gi←cyli metrici∘cyl
   gi ip g 3 1 4
1.00001        0.00000905151 0
¯0.00000381988 0.999993      0
 0             0             1

   g ip gi 3 1 4
1.00001       ¯0.00000381988 0
0.00000905151  0.999993      0
0              0             1
```

Raising and lowering indices

In Einstein notation, the fundamental tensors can be used to "raise and lower" indices. In other words, this means that a function producing a covariant tensor field can be modified so that it now transforms as a contravariant tensor; and *vice versa*.

$A^i = g^{ij} A_j$	*Covariant to contravariant*
$A_i = g_{ij} A^j$	*Contravariant to covariant*

We'll demonstrate this with two examples, using the following definitions:

```
   trf←{3 1 4××ω} ⋄ trfi←{⍟ω÷3 1 4}
   Q←trf P←3 1 4
   g←trf metric
   gi←trfi metrici∘trf
   T←trf ∆ ⋄ TI←trfi ∆
   f←{ω*3}
```

First, if `f` is used to define a covariant vector field, does `gi ip f` transform as a contravariant tensor?

```
   t←('i.j-.-'val T op T op gi∘trfi Q) +.× '.i.'val TI op(f∘trfi)Q
   t comp'i..'val T op(gi ip f)∘trfi Q
1
```

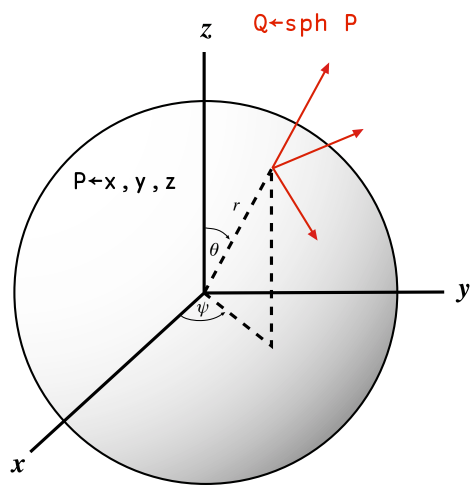
Yes, it does. Now let's consider `f` as forming a contravariant vector field. Does `g ip f`, transform as a contravariant tensor?

```
   j←('.i-j.-'val TI op TI op(g∘trfi)Q) +.× 'i..'val T op f∘trfi Q
   j comp'.i.'val TI op(g ip f∘trfi)Q
1
```

It is interesting to note that the function we've defined here `f` has remained unchanged throughout both examples. It was employed to create a field and the values provided only depended on the coordinates `Q`, not on the transformation function `trf`. Separately, we declared whether the field was covariant or contravariant. Sokolnikoff refers to tensors derived by raising and lowering indices in this way as *associated* tensors.

Testing the metric's tensor character

Earlier we described how the surface of a sphere is defined as a collection of points in a `3` dimensional Cartesian reference frame. For a point `p` with coordinates `x`, `y` and `z` to lie on the surface of a sphere of radius `r` centred on the origin, its coordinates must satisfy `r=össq x,y,z`.



Points on the surface of the sphere can be identified by either their Cartesian coordinates `x`, `y` and `z` in the P space or by their spherical coordinates `r`, `theta` and `psi` in the Q space and we can switch between the two with:

```
   r theta psi←sph x,y,z
```

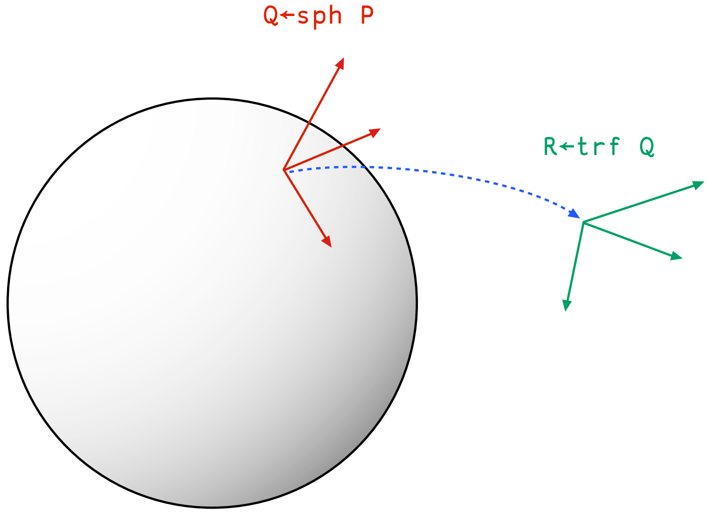
and

```
   x y z←sphi r,theta,psi
```

At a point `P` on the surface, we can identify local axes (shown in red) corresponding to increasing values of the spherical coordinates r, θ and ψ. Using those axes, we can define points with coordinates relative to `P`. And from there we can then define vectors as the difference between pairs of those points.

One vector of particular interest is the differential element which we have used as the prototype for a contravariant tensor. On the surface, in the coordinates of the Q space, the differential element is given by `diff Q`. The metric tensor on the surface is calculated from `sph`, the function that transforms coordinates from Cartesian to spherical. Now we can calculate the square of the distance with:

```
   Q←sph 3 4 12                  Spherical coordinates
   g←sph metric                  The metric tensor for the surface
   diff ip g ip diff Q           The square of the differential element
1.70016E¯10
```



Let's now consider how this appears to a different observer (in the R space), one whose coordinates are related to the Q space by a transform function `trf`.

```
   trf←{3 1 4××ω} ⋄ trfi←{⍟ω÷3 1 4}
   R←trf Q                                  from Q space to R space
   T←trf ∆ ⋄ TI←trfi ∆
   diffr←T ip diff∘trfi                     As diff is a contravariant vector tensor
   gr←{'.-.i-j'val g∘trfi op TI op TI ω}    As g is a covariant matrix tensor
   diffr ip gr ip diffr R
1.70016E¯10
```

This demonstrates that the metric transforms as a covariant matrix tensor.

The Covariant Derivative Operator

The covariant derivative is rather special. It is an essential tool for writing the equations of physics in a frame invariant form – that is as tensor equations. It is not the same as the regular derivative.

There are several approaches to establishing a definition. Dirac and many others appeal to a geometric approach, relying on understanding how the parallel transport of a vector in a curved space takes place. I've chosen to follow Sokolnikoff and use a more analytical approach.

However, both approaches involve an understanding of the Christoffel symbols.

Christoffel symbols

Christoffel symbols are rank `3` functions of the metric tensor. There are two kinds.

The symbol of the first kind `C1` is defined in terms of the derivative of the metric tensor of a surface produced by a function `trf` as:

```
chr←{0.5×(0 2 1⍉ω)+(1 2 0⍉ω)-ω}
g←trf metric
gi←trfi metrici∘trf
C1←chr g ∆   ..................................................................... [14]
```

Sokolnikoff defines the symbol of the second kind as:

```
C2←'i.jk.'val gi op C1
```

We'll use a slightly different, but equivalent, expression:

```
   C2←C1 ip ⍉gi
»  C1 ip gi   ..................................................................... [15]
```

It is clear from the definitions that `C1` and `C2` are both symmetric with respect to the first two axes. Let's show this with an example:

```
   trf←sph ⋄ trfi←sphi
   Q←trf P←3 4 12
   g←(⍉dsph)ip dsph                            For better numerical accuracy
   gi←dsphi ip(⍉dsphi)∘sph                     ditto
   C1←chr g ∆
   C2←C1 ip gi
   {ω comp 1 0 2⍉ω}C1 Q
1
   {ω comp 1 0 2⍉ω}C2 Q
1
```

Note that `C1` can be expressed in terms of `C2` with:

```
   C1  ↔  C2 ip g   ................................................................ [16]
```

The derivative of the metric tensor

We can express the derivative of the metric tensor in terms of `C1`. Consider the expression:

```
('ikj'val C1 Q)+'jki'val C1 Q
```

Writing `m←g ∆ Q` and substituting for `C1 Q` as `chr m`, we have:

```
»  (0 2 1⍉chr m)+1 2 0⍉chr m
»  (0 2 1⍉0.5×(0 2 1⍉m)+(1 2 0⍉m)-m)+1 2 0⍉0.5×(0 2 1⍉m)+(1 2 0⍉m)-m
»  0.5×(0 2 1⍉0 2 1⍉m)+(0 2 1⍉1 2 0⍉m)+(-0 2 1⍉m)
      +(1 2 0⍉0 2 1⍉m)+(1 2 0⍉1 2 0⍉m)+(-1 2 0⍉m)
»  0.5×m+(2 1 0⍉m)+(-0 2 1⍉m)+(1 0 2⍉m)+(2 0 1⍉m)+(-1 2 0⍉m)
```

Because of the symmetry of the first two axes of `m`, the second and sixth terms cancel; as do the third and fifth terms. This leaves:

```
»  0.5×m+1 0 2⍉m
```

Again, because of the symmetry, these two terms are identical and the expression reduces to `m`. Therefore:

```
trf metric ∆ Q  ↔  {('ikj'val ⍵)+'jki'val ⍵}C1 Q  ............................  [17]
```

Transformation of the Christoffel symbols

Are the Christoffel symbols tensors? To answer this question we need to look at what we get if we transform the component parts of `C1` (or `C2`). If, having done those transformations, we end up with an expression that is just a tensor transformation of `C1`, then we are done: `C1` is a tensor. Otherwise, not.

Transformation of `C1`

`C1` is defined in terms of the metric tensor `g` for a space. As `g` is a covariant tensor of rank `2`, it transforms to be in the R space:

```
h←'.i-j.-'val TI op TI op(g∘trfi)
```

We can construct the Christoffel symbol of the first kind for the transformed metric `h`. This is:

```
D1←chr h ∆
```

To expand this, so we can get back to `g`, will be done in two steps. First we'll expand the derivative of `h` in terms of `g`. Then we'll expand the effect of `chr` applied to `h ∆ R`.

To simplify some of the typing let's define:

```
a←TI op TI op(g∘trfi ∆)R
b←TI op(TI ∆)op(g∘trfi)R
c←(TI ∆)op TI op(g∘trfi)R
t←T Q
ti←TI R
```

Note that:

```
b≡2 3 4 0 1 5 6⍉c  ....................................................  [18]
```

(1) The derivative of `h`

```
   h ∆ R
»  ('.i-j.-'val TI op TI op(g∘trfi))∆ R
```

We can bring the `val` function outside the derivative:

```
»  '.i-j.-k'val(TI op TI op(g∘trfi))∆ R
```

Expanding the derivative of the rightmost outer product with `TI`, we get:

```
»  '.i-j.-k'val a+0 1 2 3 6 4 5⍉TI op TI ∆ op(g∘trfi)R
»  ('.i-j.-k'val a)+'.i-j.-k'val 0 1 2 3 6 4 5⍉TI op TI ∆ op(g∘trfi)R
```

We can remove the dyadic transform in the second term by combining it with the `val` function:

```
»  ('.i-j.-k'val a)+'.i-jk.-'val TI op TI ∆ op(g∘trfi)R
```

Expanding the `TI op TI ∆` term, we have:

```
»  ('.i-j.-k'val a)+'.i-jk.-'val((TI op(TI ∆))+0 1 4 2 3⍉(TI ∆ op TI))op(g∘trfi)R
»  ('.i-j.-k'val a)+'.i-jk.-'val b+0 1 4 2 3 5 6⍉c
»  ('.i-j.-k'val a)+('.i-jk.-'val b)+'.i-jk.-'val 0 1 4 2 3 5 6⍉c
```

Again, we can combine the dyadic transform of `c` with the `val` function in the last term:

```
»  ('.i-j.-k'val a)+('.i-jk.-'val b)+'.ik-j.-'val c
```

The central term here manipulates the array `b`. We can revise this using [18] to instead work on the array `c`:

```
»  ('.i-j.-k'val a)+('.i-jk.-'val 2 3 4 0 1 5 6⍉c)+'.ik-j.-'val c
```

and

```
   h ∆ R  ↔  ('.i-j.-k'val a)+('-jk.i.-'val c)+'.ik-j.-'val c  ....................  [19]
```

(2) The Christoffel symbol for `h`

Now let's return to the Christoffel symbol of the first kind for the transformed metric `h`. This is:

```
   D1←chr h ∆ R
»  (chr'.i-j.-k'val a)+(chr'-jk.i.-'val c)+chr'.ik-j.-'val c
```

We can expand `chr` in the last two terms to get:

```
»  (chr'.i-j.-k'val a)
   +0.5×('ikj'val'-jk.i.-'val c)+('jki'val'-jk.i.-'val c)+(-'-jk.i.-'val c)
       +('ikj'val'.ik-j.-'val c)+('jki'val'.ik-j.-'val c)+(-'.ik-j.-'val c)
```

The last six terms all refer to `c` and, of these, four have a double application of `val`. We can simplify using `merge`, as follows:

```
   ix←(6⍴'ikj' 'jki' 'ijk')merge¨3/'-jk.i.-' '.ik-j.-'
   c ix¨ix
```

.kj-i-.	.ki-j-.	.jk-i-.	-ij.k-.	-ji.k-.	-ik.j-.

```
   d←ix val¨⊂c

»  (chr'.i-j.-k'val a)+⊃+/0.5×1 1 ¯1 1 1 ¯1×d
```

In this expression `d` has six elements, each being derived from `c`. Due to symmetry, a number of these terms are the same, and we can simplify. We can see where the matches are with:

```
   1=d∘.comp d
1 0 1 0 0 0
0 1 0 0 0 1
1 0 1 0 0 0
0 0 0 1 1 0
0 0 0 1 1 0
0 1 0 0 0 1
```

As the first and third terms match and they are of opposite sign, they cancel out. Similarly for the second and the sixth terms. We are just left with the fourth and fifth terms. These two terms are identical and their sum eliminates the `0.5` factor. This gives:

```
»  (chr'.i-j.-k'val a)+'-ij.k-.'val c
```

Replacing `a` in the first term by `TI op TI op(g∘trfi ∆)r` and evaluating the derivative of the `g∘trfi` composition:

```
»  (chr'.i-j.-k'val TI op TI op(g∘trfi ∆)R)+'-ij.k-.'val c
»  (chr'.i-j.-k'val ti∘.×ti∘.×g∘trfi ∆ R)+'-ij.k-.'val c
»  (chr'.i-j.-k'val ti∘.×ti∘.×(g ∆ trfi R)+.×ti)+'-ij.k-.'val c
»  (chr'.i-j*k.-*'val ti∘.×ti∘.×ti∘.×(g ∆ q))+'-ij.k-.'val c
»  ('.i-j*k.-*'val ti∘.×ti∘.×ti∘.×chr g ∆ q)+'-ij.k-.'val c
»  ('.i-j*k.-*'val ti∘.×ti∘.×ti∘.×C1 q)  +  '-ij.k-.'val c
```

The first term in this expression above is just the transformed version of `C1`, the type `1` Christoffel symbol for the metric tensor `g`. It is interesting to compare this with the comparable expression shown by Sokolnikoff (equation 32.3 at p. 83) in Einstein notation:

$$\partial x^{\alpha}/\partial y^{i}\ \partial x^{\beta}/\partial y^{j}\ \partial x^{\gamma}/\partial y^{k}\ {}_{x}[\alpha\beta,\gamma]$$

As the Christoffel function `C1` is of rank `3`, `TI` appears three times as we'd expect. But what about the index argument to `val`? Do we have indices that correspond with those used in the Einstein notation? Here's how to check. Simply write out the indices as they appear in Einstein notation and apply the `cix` function:

```
   cix 'αiβjγkαβγ'
.i-j*k.-*
```

If we could stop there, we would conclude that `C1` is a tensor. However, there is that second term, which is not zero in general. The conclusion must be that `C1` is not a tensor.

The second term is shown by Sokolnikoff in Einstein notation as:

$$\partial^{2}x^{\alpha}/\partial y^{i}\ \partial y^{j}\ \partial x^{\beta}/\partial y^{k}\ g_{\alpha\beta}$$

Does this match the APL term `'-ij.k-.'val c`? Let's expand `c` and see what we have:

```
   '-ij.k-.'val c
»  '-ij.k-.'val(TI ∆)op TI op(g∘trfi)r
```

We clearly have a `TI ∆` term, which is the second derivative of `trfi`; and a term for the transform for the covariant `g`. Do we have the correct indices to use with `val`?

```
   '-ij.k-.' ≡⍥cix 'αijβkαβ'
1
```

In summary, `C1` transforms to become:

```
   D1←('.i-j*k.-*'val TI op TI op TI op(C1∘trfi)R)
      +'-ij.k-.'val(TI ∆)op TI op(g∘trfi)R   ...................................... [20]
```

Transformation of `C2`

We can determine how `C2` transforms from its definition as `C1 ip gi`. It will be `D2←D1 ip hi` where:

```
   hi←'i.j-.-'val T op T op gi∘trfi

   D1 ip hi R
»  (('.i-j*k.-*'val TI op TI op TI op(C1∘trfi)R)+.×hi R)
   +
   ('-ij.k-.'val(TI ∆)op TI op(g∘trfi)R)+.×hi R
```

Let's deal with these two terms one at a time. For the first term we have:

```
   ('.i-j*k.-*'val TI op TI op TI op(C1∘trfi)R)+.×hi R
»  ('.i-j*k.-*'val ti∘.×ti∘.×ti∘.×C1 Q)+.×hi r
»  ('.i-j*k.-*'val ti∘.×ti∘.×ti∘.×C1 Q)+.×'i.j-.-'val T op T op gi∘trfi R
»  ('.i-j*∘.-*'val ti∘.×ti∘.×ti∘.×C1 Q)+.×'∘=k+=+'val T op T op gi Q
»  '.i-j*∘.-*∘=k+=+'val ti∘.×ti∘.×ti∘.×(C1 Q)∘.×T op T op gi Q
»  '.i-j.-**∘∘=k+=+'val ti∘.×ti∘.×(C1 Q)∘.×ti∘.×T op T op gi Q
»  '.i-j.-**=k+=+'val ti∘.×ti∘.×(C1 Q)∘.×ti+.×T op T op gi Q
»  '.i-j.-=k+=+'val ti∘.×ti∘.×(C1 Q)+.×(idρQ)∘.×T op gi Q
»  '.i-j.-=k+=+'val ti∘.×ti∘.×(C1 Q)∘.×T op gi Q
»  '.i-jk+.-==+'val ti∘.×ti∘.×t∘.×(C1 Q)∘.×gi Q
»  '.i-jk+.-+'val ti∘.×ti∘.×t∘.×C1 ip gi Q
»  '.i-jk+.-+'val ti∘.×ti∘.×t∘.×C2 Q

»  'k+.i-j.-+'val t∘.×ti∘.×ti∘.×C2 Q   ........................................ [21]
```

Sokolnikoff shows the equivalent in Einstein notation as:

$\partial y^k/\partial x^\varrho\ \partial x^\alpha/\partial y^i\ \partial x^\beta/\partial y^j\ {}_x\{\varrho,\alpha\beta\}$

(Note that this is where Einstein notation starts to become awkward. The subscripted x in front of the Christoffel symbol indicates "*to be evaluated in the untransformed space*". In APL, this is simply accomplished by choosing `Q` as the function argument.)

This is just what we should expect for the transformation of `C2` if it were a mixed tensor, twice covariant and once contravariant. However, because of the second term, `C2` is not a tensor (unless `trf` is affine).

The second term is:

```
   ('-ij.k-.'val c)+.×hi R
»  ('-ij.k-.'val(TI ∆)op TI op(g∘trfi)R)+.×'i.j-.-'val T op T op gi∘trfi R
»  ('-ij.k-.'val(TI ∆ R)∘.×ti∘.×(g Q))+.×'i.j-.-'val t∘.×t∘.×gi Q
»  ('-ij.∘-.'val(TI ∆ R)∘.×ti∘.×(g Q))+.×'∘*k=*='val t∘.×t∘.×gi Q
»  '-ij.∘-.∘*k=*='val(TI ∆ R)∘.×ti∘.×(g Q)∘.×t∘.×t∘.×gi Q
»  '-ij-..∘∘*k=*='val(TI ∆ R)∘.×(g Q)∘.×ti∘.×t∘.×t∘.×gi Q
»  '-ij-..*k=*='val(TI ∆ R)∘.×(g Q)∘.×ti+.×t∘.×t∘.×gi Q
»  '-ij-..*k=*='val(TI ∆ R)∘.×(g Q)∘.×(idρQ)∘.×t∘.×gi Q
»  '-ij-*k=*='val(TI ∆ R)∘.×(g Q)∘.×t∘.×gi Q
»  '-ijk=-**='val(TI ∆ R)∘.×t∘.×(g Q)∘.×gi Q
»  '-ijk=-='val(TI ∆ R)∘.×t∘.×idρ Q
»  '-ijk==-'val(TI ∆ R)∘.×t∘.×idρ Q
»  '-ijk-'val(TI ∆ R)∘.×t+.×idρ Q

»  '-ijk-'val(TI ∆ R)∘.×t   .................................................. [22]
```

Sokolnikoff shows the equivalent in Einstein notation as:

$\partial^2 x^\alpha / \partial y^i \, \partial y^j \, \partial y^k / \partial x^\alpha$

Recombining the two terms we have:

```
   D2 R
»  ('k+.i-j.-+'val to.×tio.×tio.×C2 Q)+'-ijk-'val(TI ∆ R)o.×t
»  ('k+.i-j.-+'val to.×tio.×tio.×C2 Q)+'kij'val t+.×TI ∆ R
```

We can solve this equation for `TI ∆ R`. The first step is to apply a `1 2 0⍉` to all the terms. Doing so, we have:

```
   1 2 0⍉D2 R
»  (1 2 0⍉'k+.i-j.-+'val to.×tio.×tio.×C2 Q)+t+.×TI ∆ R
```

Then we apply a left inner product with `ti`:

```
   ti+.×1 2 0⍉D2 R
»  (ti+.×1 2 0⍉'k+.i-j.-+'val to.×tio.×tio.×C2 Q)+TI ∆ R
```

Rearranging terms, this gives:

```
   TI ∆ R
»  (ti+.×1 2 0⍉D2 R)-ti+.×1 2 0⍉'k+.i-j.-+'val to.×tio.×tio.×C2 Q
»  (ti+.×1 2 0⍉D2 R)-ti+.×'i+.j-k.-+'val to.×tio.×tio.×C2 Q
»  (ti+.×1 2 0⍉D2 R)-'i+.j-k.-+'val ti+.×to.×tio.×tio.×C2 Q
»  (ti+.×1 2 0⍉D2 R)-'i+.j-k.-+'val(idρQ)o.×tio.×tio.×C2 Q
»  (ti+.×1 2 0⍉D2 R)-'.j-k.-+i+'val tio.×tio.×(C2 Q)o.×idρQ
»  (ti+.×1 2 0⍉D2 R)-'.j-k.-++i'val tio.×tio.×(C2 Q)o.×idρQ
»  (ti+.×1 2 0⍉D2 R)-'.j-k.-i'val tio.×tio.×C2 Q
»  (ti+.×'jki'val D2 R)-'.j-k.-i'val tio.×tio.×C2 Q
»  ('i.jk.'val tio.×D2 R)-'.j-k.-i'val tio.×tio.×C2 Q   ............................ [23]
```

Covariant derivative of a covariant vector field

Consider the field produced by a covariant vector function `f`. As it's covariant it transforms to be `F←(f∘trfi)ip TI`. What is the derivative of `F`?

We'll continue using the example of working on the surface of a sphere. Our sample point will still be `Q` and we'll consider a vector field `f` being transformed by the twisted function `tw` to `F` in the `R` space. For example:

```
   f←{ω*3}
```

We can form `F`'s derivative in the usual way:

```
   F ∆ R
»  (f∘trfi)ip TI ∆ R
»  ((f Q)+.×TI ∆ R)+¯1 sh(1 sh(f∘trfi)∆ R)+.×TI R
»  ((f Q)+.×TI ∆ R)+⍉(⍉(f ∆ trfi R)+.×ti)+.×ti
»  ((f Q)+.×TI ∆ R)+⍉(⍉(f ∆ Q)+.×ti)+.×ti
»  ((f Q)+.×TI ∆ R)+⍉(⍉'i..j'val(f ∆ Q)o.×ti)+.×ti
»  ((f Q)+.×TI ∆ R)+⍉('j..i'val(f ∆ Q)o.×ti)+.×ti
»  ((f Q)+.×TI ∆ R)+⍉'j..ijk'val(f ∆ Q)o.×tio.×ti
»  ((f Q)+.×TI ∆ R)+'-..j-i'val(f ∆ q)o.×tio.×ti
```

Now we can replace `TI ∆ R` by its value from equation [23] above giving:

```
»  ((f Q)+.×('i.jk.'val tio.×D2 R)-'.j-k.-i'val tio.×tio.×C2 Q)+'-..j-i'val(f ∆ Q)o.×tio.×ti
```

As this expression is getting long, we'll label the three component parts and work on them individually:

```
    x←(f Q)+.×'i.jk.'val ti∘.×D2 R
    y←(f Q)+.×'.j-k.-i'val ti∘.×ti∘.×C2 Q
    z←'-..j-i'''val(f ∆ Q)∘.×ti∘.×ti
```

Just to confirm:

```
    ((F ∆ R)+y)comp x+z
1
```

We can simplify `x`, `y` and `z` as:

```
    x
»   'ii.jk.'val(f q)∘.×ti∘.×D2 r
»   '.jk.'val(F R)∘.×D2 R
»   'kj'val(F R)+.×⍉D2 R
»   (D2 R)+.×F R

    y
»   '+.i-j.-+'val(f Q)∘.×ti∘.×ti∘.×C2 Q
»   '.i-j.-++'val ti∘.×ti∘.×(C2 Q)∘.×f Q
»   '.i-j.-'val ti∘.×ti∘.×(C2 Q)+.×f Q
    z
»   '.j-i-.'val ti∘.×ti∘.×f ∆ Q
»   '.i-j.-'val ti∘.×ti∘.×f ∆ Q
```

And we can rewrite the equivalence as:

```
    (F ∆ R)-(D2 R)+.×F R  ←→  '.i-j.-'val ti∘.×ti∘.×(f ∆ Q)-(C2 Q)+.×f Q
```

which shows that `(f ∆ Q)-(C2 Q)+.×f Q` transforms as a rank `2` covariant tensor. This forms the basis for the definition of the covariant derivative operator for a covariant vector field:

```
    ∆cov←{(⍺⍺ ∆ ⍵)+(-C2)ip ⍺⍺ ⍵}  ............................................. [24]
```

Covariant derivative of a contravariant vector field

What if we had begun with a contravariant vector function `f`. What would its derivative be?

As `f` is contravariant it transforms to be `F←T ip f∘trfi` which we can differentiate:

```
    F ∆ R
»   (T∘trfi)ip(f∘trfi)∆ R
»   ((T∘trfi)ip(f∘trfi ∆)r)+¯1 sh(1 sh(T∘trfi ∆)r)+.×f∘trfi r
»   (t+.×f∘trfi ∆ R)+¯1 sh(1 sh(T∘trfi ∆)r)+.×f Q
»   (t+.×f∘trfi ∆ R)+¯1 sh(1 sh(T∘trfi ∆)r)+.×f Q
»   (t+.×f∘trfi ∆ R)+¯1 sh(1 sh(T∘trfi ∆)r)+.×f Q
»   (t+.×f∘trfi ∆ R)+¯1 sh(1 sh(T ∆ Q)+.×ti)+.×f Q
»   (t+.×f∘trfi ∆ R)+¯1 sh('jki'val(T ∆ Q)+.×ti)+.×f Q
»   (t+.×f∘trfi ∆ R)+¯1 sh('jk..i'val(T ∆ Q)∘.×ti)+.×f Q
»   (t+.×f∘trfi ∆ R)+¯1 sh'j-..i-'val(T ∆ Q)∘.×ti∘.×f Q
»   (t+.×f∘trfi ∆ R)+'i-..j-'val(T ∆ Q)∘.×ti∘.×f Q
»   (t+.×(f ∆ Q)+.×ti)+'i-..j-'val(T ∆ Q)∘.×ti∘.×f Q
```

We can replace `T ∆ Q` with its equivalent `-'*-.i*.k-j'val(TI ∆ R)∘.×t∘.×t∘.×t` from equation [5] above:

```
»  (t+.×(f ∆ Q)+.×ti)-'i-..j-'val('*-.i*.k-j'val(TI ∆ R)∘.×t∘.×t∘.×t)∘.×ti∘.×f Q
»  (t+.×(f ∆ Q)+.×ti)-'i-..j-'val'*-.i*.k-jlmn'val(TI ∆ R)∘.×t∘.×t∘.×t∘.×ti∘.×f Q
```

Now we can combine the two applications of `val`:

```
   cix'i-..j-'merge'*-.i*.k-jlmn'
.-*i.*=-∘=j∘
```

```
»  (t+.×(f ∆ Q)+.×ti)-'.-*i.*=-∘=j∘'val(TI ∆ R)∘.×t∘.×t∘.×t∘.×ti∘.×f Q
»  (t+.×(f ∆ Q)+.×ti)-'i..-**==j-∘∘'val t∘.×(TI ∆ R)∘.×t∘.×ti∘.×t∘.×f Q
»  (t+.×(f ∆ Q)+.×ti)-'i-j-'val t+.×(TI ∆ R)+.×t+.×ti∘.×t+.×f Q
»  (t+.×(f ∆ Q)+.×ti)-'i-j-'val t+.×(TI ∆ R)∘.×F R
```

Substituting `TI ∆ R` as `('i.jk.'val ti∘.×D2 R)-'.j-k.-i'val ti∘.×ti∘.×C2 Q` (from equation [23] above), we have:

```
»  (t+.×(f ∆ Q)+.×ti)
   -('i-j-'val t+.×('i.jk.'val ti∘.×D2 r)∘.×F R)
   -'i-j-'val t+.×('.j-k.-i'val ti∘.×ti∘.×C2 q)∘.×F R
```

As before, for brevity, we'll label the three component parts and work on them individually:

```
   x←t+.×(f ∆ Q)+.×ti
   y←'i-j-'val t+.×('i.jk.'val ti∘.×D2 R)∘.×F R
   z←'i-j-'val t+.×('.j-k.-i'val ti∘.×ti∘.×C2 Q)∘.×F R
```

We can simplify `y` and `z` as:

```
   y
»  'i-j-'val t+.×'i.jk.l'val ti∘.×(D2 R)∘.×F R
»  'i-j-'val'i.jk.l'val t+.×ti∘.×(D2 R)∘.×F R
»  'i-j-'val'i.jk.l'val(idρR)∘.×(D2 R)∘.×F R
»  'i.-j.-'val(idρR)∘.×(D2 R)∘.×F R
»  'i..j--'val(idρR)∘.×(⍉D2 R)∘.×F R
»  (⍉D2 R)+.×F R

   z
»  'i-j-'val t+.×'.j-k.-il'val ti∘.×ti∘.×(C2 Q)∘.×F R
»  'i-j-'val t+.×'.j-ki-.l'val ti∘.×ti∘.×(⍉C2 Q)∘.×F R
»  'i-j-'val'i=.j-k=-.l'val t∘.×ti∘.×ti∘.×(⍉C2 Q)∘.×F R
»  'i*-∘.j*.-∘'val t∘.×ti∘.×ti∘.×(⍉C2 Q)∘.×F R
»  'i*.j*.'val t∘.×ti∘.×(⍉C2 Q)+.×ti+.×F R
»  'i*.j*.'val t∘.×ti∘.×(⍉C2 Q)+.×f Q
»  'i**.--.j'val t∘.×(⍉C2 Q)∘.×(f Q)∘.×t
»  t+.×((⍉C2 Q)+.×f Q)+.×ti
```

Now we have for the equivalence:

```
   (F ∆ R)+(⍉D2 R)+.×F R  ↔  t+.×((f ∆ Q)+(⍉C2 Q)+.×f Q)+.×ti
```

which shows that `(f ∆ Q)+(⍉C2 Q)+.×f Q` transforms as a rank `2` mixed tensor. This forms the basis for the definition of the covariant derivative operator for a contravariant field:

```
   ∆con←{(⍺⍺ ∆ ⍵)+(⍉C2)ip ⍺⍺ ⍵}  ............................................ [25]
```

Covariant derivatives of matrix fields

Naturally, once we have an expression for the covariant derivative of a covariant vector field, we'll need to see what this sort of analysis produces for the covariant derivative of a covariant matrix field. We could steel ourselves and work through the analysis that produced equations [23] and [24], but this would be some labour. Fortunately there is another way.

We start with the observation that a matrix field `M` can be formed as the sum of terms of the form `u op v`, where `u` and `v` are vector functions (Dirac p.18):

```
M←(u0∘.×v0)+(u1∘.×v1) ....
```

And our starting point can then be `(u op v)∆cov`.

In order to expand this we'll assume that the following identity holds (relying on Dirac p. 18 who defines it that way):

```
(u up v)∆cov Q  ←→  (u op(v ∆cov)Q)+0 2 1⍉(u ∆cov)op v Q
```

We'll keep things brief and hopefully clearer with some simple defintions:

```
a←u q ⋄ b←v q ⋄ c←u ∆ q ⋄ d←v ∆ q ⋄ e←C2 q ⋄ uv←u op v
```

Then we have:

```
   (u op(v ∆cov)Q)+0 2 1⍉(u ∆cov)op v Q
»  (a∘.×d-e+.×b)+0 2 1⍉(c-e+.×a)∘.×b
»  (uv ∆ Q)-(a∘.×e+.×b)+0 2 1⍉(e+.×a)∘.×b
»  (uv ∆ Q)-('ijk∘∘'val a∘.×e∘.×b)+0 2 1⍉'ij∘∘k'val e∘.×a∘.×b
»  (uv ∆ Q)-{('jk∘i∘'val ⍵)+'ik∘∘j'val ⍵}e∘.×a∘.×b

»  (uv ∆ Q)+{('jk∘i∘'val ⍵)+'ik∘∘j'val ⍵}(-C2)op uv Q  ............................  [26]
```

We can do the same for the covariant derivative of an entirely contravariant matrix field producing:

```
   (u op(v ∆con)Q)+0 2 1⍉(u ∆con)op v Q

»  (uv ∆ Q)+{('jk∘i∘'val ⍵)+'ik∘∘j'val ⍵}(⍉C2)op uv Q  ............................  [27]
```

Mixed variance matrix fields come in two types – one with the two axes being covariant followed by contravariant, and the other with the variances interchanged. Their covariant derivatives are:

```
   (uv ∆ Q)+{('ik∘∘j'val(-C2)op uv Q)+'jk∘i∘'val ⍵}(⍉C2)op uv Q  ..................  [28]

   (uv ∆ Q)+{('ik∘∘j'val(⍉C2)op uv Q)+'jk∘i∘'val ⍵}(-C2)op uv Q  ..................  [29]
```

Notice that the terms adjusting the ordinary derivative `uv ∆ Q` apply the same `val` transformations, but just to different arguments. For a covariant axis, the argument contains `-C2`; for a contravariant axis, the argument contains `⍉C2`.

Of course, this is just the lead up to dealing with a definition for the covariant derivative able to take on any field.

The general covariant derivative

First we'll need a general way to form the index vectors that appear as left arguments to `val`. We can use:

```
ix←{a←(⍳⍵),(⍵,2)⍴⍵+0 1                        Index generator
  b←id ⍵
  ↓a,(b×⍵+1)+(~b)×(⍵,⍵)⍴⍳⍵}
```

This takes an argument of the rank of the field in question. Here are two examples:

```
   cix¨ix 2
┌────┬────┐
│ik..j│jk.i.│
└────┴────┘
   cix¨ix 3
┌─────┬─────┬─────┐
│il..jk│jl.i.k│kl.ij.│
└─────┴─────┴─────┘
```

Now we have all we need to put together an operator ⍙ that can handle any field. But, we will have to specify which axes are covariant and which are contravariant – and that we'll do in a left argument. Here's the definition for ⍙:

```
   ⍙←{t←αα ω
     c←C2 ω ⋄ c←(⊂-c∘.×t),⊂(⍉c)∘.×t
     (αα ∆ ω)+⊃+/(ixρα)val¨c[α]}   ........................................  [30]
```

⍙ is a dyadic operator. Its application to a function left argument produces a dyadic function. The left argument to that derived function is a boolean indicating whether each axis is covariant `(=0)` or contravariant `(=1)`; the right argument is just a vector coordinates.

For comparison, here's the same result in Einstein notation (Sokolnikoff p. 86):

$$A^{j_1\cdots j_s}_{i_1\cdots i_r,l} \equiv \frac{\partial A^{j_1\cdots j_s}_{i_1\cdots i_r}}{\partial x^l}$$

$$(33.5) \qquad - \begin{Bmatrix} \alpha \\ i_1 l \end{Bmatrix} A^{j_1\cdots j_s}_{\alpha i_2\cdots i_r} - \begin{Bmatrix} \alpha \\ i_2 l \end{Bmatrix} A^{j_1\cdots j_s}_{i_1 \alpha i_3\cdots i_r} - \cdots - \begin{Bmatrix} \alpha \\ i_r l \end{Bmatrix} A^{j_1\cdots j_s}_{i_1\cdots \alpha}$$

$$+ \begin{Bmatrix} j_1 \\ \alpha l \end{Bmatrix} A^{\alpha j_2\cdots j_s}_{i_1\cdots i_r} + \begin{Bmatrix} j_2 \\ \alpha l \end{Bmatrix} A^{j_1 \alpha j_3\cdots j_s}_{i_1\cdots i_r} + \cdots + \begin{Bmatrix} j_s \\ \alpha l \end{Bmatrix} A^{j_1\cdots \alpha}_{i_1\cdots i_r}.$$

Note that Sokolnikoff's version does not reveal the order in which the `r+s` indices should appear. This puts the result in doubt as there is no indication how to apply the necessary adjusting dyadic transforms.

Examples

We'll continue with our example at a point Q on the surface of a sphere. The relevant definitions for that are:

```
   Q←sph P←3 4 12
   g←(⍉dsph)ip dsph
   gi←dsphi ip(⍉dsphi)∘sph
   C1←chr g ∆
   C2←C1 ip gi
```

On the surface of this sphere, we'll define some functions to produce fields:

```
u←{⍵*0.9 1 1.2}                                   Vector field
v←{3 1 4+⍵×2 7 1}                                 ditto
uv←u op v                                         Matrix field
phi←u ip v                                        Scalar field
psi←{+/⍵}                                         Another scalar field
```

First we'll check that our definition for ⍙ does produce the correct value for its application to a covariant field:

```
   (0 u ⍙ Q)comp(u ∆ Q)-C2 ip u Q
1
```

We can do the same test, treating `u` as the associated contravariant field:

```
   (1 u ⍙ Q)comp(u ∆ Q)+(C2 ip u)Q
1
```

And then for the scalar field `phi`:

```
   (⍬ phi ⍙ Q)comp phi ∆ Q
1
```

For the matrix field `uv` there are four possibilities for the variance quality of the field: purely covariant (`0 0`), mixed (either `0 1` or `1 0`) or purely contravariant (`1 1`). These check with:

```
   (0 0 uv ⍙ Q)comp(uv ∆ Q)+{('ik∘∘j'val ⍵)+'jk∘i∘'val ⍵}(-C2)op uv Q

1
   (0 1 uv ⍙ Q)comp(uv ∆ Q)+('ik∘∘j'val(-C2)op uv Q)+'jk∘i∘'val(⍉C2)op uv Q

1
   (1 0 uv ⍙ Q)comp(uv ∆ Q)+('ik∘∘j'val(⍉C2)op uv Q)+'jk∘i∘'val(-C2)op uv Q

1
   (1 1 uv ⍙ Q)comp(uv ∆ Q)+{('ik∘∘j'val ⍵)+'jk∘i∘'val ⍵}(⍉C2)op uv Q

1
```

Exploring the properties of the covariant derivative

We can now test out the covariant versions of identities made with the ordinary derivative `∆`.

Addition

Let's start with the derivative of the sum of two functions. The rule for the ordinary derivative is:

```
   (u+v)∆ q  ↔  (u ∆ q)+v ∆ q
```

This carries over to the covariant derivative in the following way:

```
   (⍬(phi+psi)⍙ q)comp(⍬ phi ⍙ q)+⍬ psi ⍙ q          for a scalar field
1
   (0(u+v)⍙ q)comp (0 u ⍙ q)+0 v ⍙ q                 for a covariant vector field
1
   (1(u+v)⍙ q)comp (1 u ⍙ q)+1 v ⍙ q                 for a contravariant vector field
1
```

In general, if `var` is the vector indicating the appropriate variance quality, the rule is:

```
   var(u+v)⍙ q  ↔  (var u ⍙ q)+var v ⍙ q  .......................................  [31]
```

Multiplication

The rule for the ordinary derivative of the product of two functions is:

```
   (u×v)∆ q  ←→  ((u xp v ∆))+v xp u ∆)q
```

This holds just fine for scalar functions:

```
   (θ(phi×psi)⍙ q)comp ((phi q)xp θ psi ⍙ q)+(psi q)xp θ phi ⍙ q
1
```

but, unfortunately, not for vector functions:

```
   (0(u×v)⍙ q)comp ((u q)xp 0 v ⍙ q)+(0 u ⍙ q)xp v q
0
   (1(u×v)⍙ q)comp ((u q)xp 1 v ⍙ q)+(1 u ⍙ q)xp v q
0.111111
```

Composition

The rule for the ordinary derivative of a composition (the chain rule) is:

```
   ((u∘v)∆ q)  ←→  (u ∆ v q)+.×v ∆ q
```

But, no such luck with the comparable equation for a covariant derivative, unless the function is scalar:

```
   (θ(phi∘u)⍙ q)comp(θ phi ⍙ u q)+.×u ∆ q
1
   (0(u∘v)⍙ q)comp (0 u ⍙ v q)+.×v ∆ q
0
```

The metric tensor

The metric tensor behaves like a constant when derivatives are taken. This is because the derivative of the metric tensor is zero. For example:

```
   ⍪disp 0 0 g ⍙ q
┌──────────────────────────────────────────────────┐
│ 4.336808690E¯19  2.602085214E¯18 ¯3.469446952E¯18│
│ 1.301042607E¯18 ¯1.387778781E¯17 ¯5.421010862E¯20│
│¯1.734723476E¯18 ¯1.626303259E¯19 ¯1.387778781E¯17│
├──────────────────────────────────────────────────┤
│ 1.301042607E¯18 ¯1.387778781E¯17 ¯5.421010862E¯20│
│ 2.168404345E¯19 ¯1.734723476E¯18 ¯1.694065895E¯21│
│¯5.421010862E¯20  1.734723476E¯18 ¯4.336808690E¯19│
├──────────────────────────────────────────────────┤
│¯1.734723476E¯18 ¯1.626303259E¯19 ¯1.387778781E¯17│
│¯5.421010862E¯20  1.734723476E¯18 ¯4.336808690E¯19│
│ 0.000000000E0   ¯1.355252716E¯20  3.469446952E¯18│
└──────────────────────────────────────────────────┘
```

This means that the metric tensor may be moved outside derivative expressions, just like the product with a constant.

Conclusion

I'm satisfied.

I set out here to get a better understanding of tensor calculus and have done so. But I only got here because I used APL. Without APL, I could have gone through the standard texts, but always would have had some doubt. Have I missed something? Glossed over something important? Of course, with APL, there do not have to be any doubts. APL is executable. I can check with actual examples.

I've always been an admirer of Einstein. He managed to explain so much, so simply. And a good part of that was his use of tensor notation. Who could argue with:

$$R^{\mu\nu} - 1/2g^{\mu\nu}R = Y^{\mu\nu}$$

for his law of gravitation in the presence of energy and matter. How he came up with this without executable confirmation, is beyond me. But he was special. He knew that the laws of physics, properly construed, had to be independent of the motion of the observer. And that lead to only one conclusion. The laws had to be expressed as tensor equations in four dimensions. For Einstein, the rest was details. Difficult work, but still details.

Of course, there are others. A favourite of mine has always been Paul Dirac. A theoretical physicist, clearly well versed in all the relevant mathematics, who achieve his fame with his theoretical formulation of quantum mechanics. However, in 1975, well after he was properly acclaimed for his work in quantum mechanics, he published a simple work with the title of "*Theory of General Relativity*". A mere 69 pages. But, it covered so much. Here's the table of contents:

This is a masterpiece. And Dirac also liked his notation terse.

And that leads me to Iverson. His contribution to the world of thought, and of notation to express that, is stunning. He stands with the masters.

I've often thought about "notation as a tool of thought". My take on this goes like this. Theories and thoughts form in unusual ways. If we want to reflect on these later, perhaps to improve them, we'll probably need to write them down. And that takes notation. If the notation facilitates that process, that's great. If the notation does more, perhaps to suggest a pattern or relationship, that is a bonus. And APL has done that for me.

Equation [30] above is a bit special for me. This provides one straightforward structure for the evaluation of the generalized covariant derivative. The dependence on variance comes down to the difference between `-C2` and `⍉C2`. And, guess what, I've never seen this before in the texts. A gift delivered by APL.

Appendix A
The Derivative Operator

Definition of the derivative operator

Here is the definition for a derivative operator `∆above`:

```
∆above←{
  k←-r←⍴n←⍴⍵
  x←(n,n)⍴⍵
  dx←(n,n)⍴(,id n)\,d←0.000001×⍵+⍵=0
  pd←(⍺⍺⍤r⊢x+dx)-⍺⍺⍤r⊢x
  pd←pd÷⍤k⊢d
  pd{(⍋((⍳⍴⍴⍺)~⍵),⍵)⍉⍺}⍳r}
```

This is known as the derivative from above as its definition calls for the limit of `(f x+dx)-f x` as `dx` approaches `0`. An alternative definition might use `(f x)-f x-dx`. This is known as the derivative from below. It's numerical approximation is slightly different and is defined as:

```
∆below←{
   k←-r←⍴n←⍴⍵
   x←(n,n)⍴⍵
   dx←(n,n)⍴(,id n)\,d←0.000001×⍵+⍵=0
   pd←(⍺⍺⍤r⊢x)-⍺⍺⍤r⊢x-dx
   pd←pd÷⍤k⊢d
   pd{(⍋((⍳⍴⍴⍺)~⍵),⍵)⍉⍺}⍳r}
```

The definition for the derivative that we'll use is just the mean of these two values:

```
∆←{0.5×(⍺⍺ ∆above ⍵)+⍺⍺ ∆below ⍵}
```

The rank of a derivative

The argument rank of the derivative of a function is just that of the function itself. This comes directly from the definition of the derivative. This has a consequence for the definition of `∆` shown above.

In order to correctly model the derivative operator, we need to know the rank of the function to which it is to be applied. This is necessary so that the shape of the data argument can be correctly broken up into its frame and cells. For example, `{⍵*2}` is a scalar function. When its derivative is applied to a vector, it should produce a vector result – as the vector should be treated as a rank `1` frame of scalars. However, observe the following:

```
   {⍵*2}∆ 2 3 4
4 0 0
0 6 0
0 0 8
```

This is incorrect. Because our definition of `∆` is not aware of the rank of its function argument, the derived function produces surplus zeros. The correct result is obtained with:

```
   {⍵*2}∆⍤0⊢2 3 4
4 6 8
```

Regrettably, Dyalog APL does not provide a means to determine the rank of a function, so, we have to define `∆` assuming that there is no frame involved, but with the caveat:

> *If the argument rank* `s` *of the function* `f` *is less than that of the argument* `x`*, then the derivative* `f ∆` *must be applied with rank* `s`*.*

Numerical accuracy of ∆

The definition of `∆` is designed to be simple to understand and generally useful as a tool for verification of expressions. However, it is a numerical approximation and it is not difficult to find examples that show up the approximation it makes.

Of particular importance in what follows, are second derivatives. In most cases, employing `∆ ∆` leads to unsatisfactory results. That's why we make use of the analytic derivatives to minimize this problem.

Appendix B
Identities & the Derivative Rules

Identities

Iverson provides a table of useful identities at p. 350. These are presented below, together with some other equivalents.

Iverson's Expression	Equivalents	Note
`f¨(g¨h)`	`(f∘g)∘h` `f∘g∘h`	Associativity of composition.

`f⊕(g⊕h)`	`(f ip g)ip h`
	`f ip g ip h`
	`((f+.×g)+.×h)`
		Associativity of the inner product operator.
		`f`, `g` and `h` must conform appropriately.

`f⊕g∘h`	`(f∘h)ip(g∘h)`
	`f ip g∘h`
	`(f+.×g)∘h`
		Composition distributes over inner product.

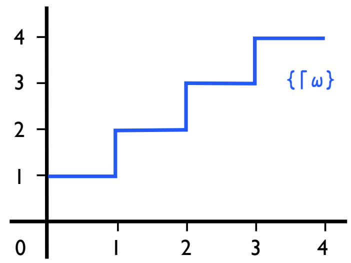
`(f⊗g)∘h`	`(f∘h)op(g∘h)`
	`f op g∘h`
	`(f∘.×g)∘h`
		Composition distributes over outer product.

`(f∘g)⍣¯1`	`(g⍣¯1)∘f⍣¯1`	Inverse of a composition. `f` and `g` are invertible rank `1 1` functions.
`g`	`f⍣¯1∘f∘g` `f∘f⍣¯1∘g`	Inverse of inverse.
`⌹¨(f¨g)`	`(⌹∘f)∘g` `⌹∘f∘g`	Associativity of composition. `f` is rank `2 1`, `g` is rank `1 1`.
`g`	`⌹∘f ip f ip g`	As `⌹∘f ip f` is an identity matrix,
`f∘g ∆`	`f ∆∘g ip(g ∆)` `(f ∆∘g+.×g ∆)`	Derivative of a composition

Differentiability

Of APL's many functions, only a few are differentiable for any argument. One example that is differentiable everywhere is the exponential function `*`.

A number of functions are only differentiable for certain domains of their argument. For example, the ceiling function `⌈`. Here's a portion of its graph:



It's clear from this that for some parts of the graph, there is a derivative. For example at `1.5`, the graph is flat with a derivative of `0`. However, at every integer value, the function is not well-defined. Consider the graph for `x←2`. It appears that the value of `⌈x` is somewhere between `2` and `3` but we can't be sure of where. And as for the gradient; it heads off to infinity – and that's a problem. Examine what happens:

```
   ceiling←{⌈⍵}
   ceiling 1.5 1.9 2 2.1 4
2 2 2 3 4
   ceiling ∆⍤0⊢ 1.5 1.9 2 2.1 4
0 0 500000 0 250000
```

Lastly, there are some functions that are not differentable at all. The deal function `{?⍵}` comes to mind.

So, bearing this in mind, let's examine the derivatives of some APL functions.

Scalar functions

Scalar functions are functions that take a scalar as an argument and return a scalar result. Their derivatives produce scalars. Here `f` and `g` are scalar functions; `s` is a scalar.

Name	Definition	Equivalent	Note
Taylor expansion	`f s+ds`	`(f s)+ds×f ∆ s`	`ds←s×0.000001`
Sum	`(f+g)∆`	`(f ∆+g ∆)s`	
Difference	`(f-g)∆ s`	`(f ∆-g ∆)s`	
Product	`(f×g)∆ s`	`((f×g ∆)+g×f ∆)s`	
Quotient	`(f÷g)∆ s`	`(((f ∆)-(f÷g)×g ∆)÷g)s`	
Composition	`(f g)∆ s`	`((f ∆ g)×g ∆)s`	
Inverse	`fi ∆ s`	`÷(f ∆ fi)s`	`fi←f⍣¯1`
Constant	`{k}∆`	`{0}`	`k←3.142`, for example
Linear	`{⍵}∆`	`{1}`	
Negate	`{-⍵}∆`	`{¯1}`	
Signum	`{×⍵}∆`	`{0}`	
Reciprocal	`{÷⍵}∆`	`{-÷⍵×⍵}`	
Power	`{⍺*⍵}∆`	`{⍵×⍺*⍵-1}`	
Exponential	`{*⍵}∆`	`{*⍵}`	
Exponential	`{⍺*⍵}∆`	`{(⍟⍺)×⍺*⍵}`	
Natural Logarithm	`{⍟⍵}∆`	`{÷⍵}`	
Sine	`{1○⍵}∆`	`{2○⍵}`	
Cosine	`{2○⍵}∆`	`{-1○⍵}`	
Tangent	`{3○⍵}∆`	`{÷(2○⍵)*2}`	
Arcsine	`{¯1○⍵}∆`	`{÷(1-⍵*2)*0.5}`	
Arccosine	`{¯2○⍵}∆`	`{-÷(1-⍵*2)*0.5}`	
Arctangent	`{¯3○⍵}∆`	`{÷1+⍵*2}`	
Hyperbolic Sine	`{5○⍵}∆`	`{6○⍵}`	
Hyperbolic Cosine	`{6○⍵}∆`	`{5○⍵}`	
Hyperbolic Tangent	`{7○⍵}∆`	`{÷(5○⍵)*2}`	
Inverse Hyperbolic Sine	`{¯5○⍵}∆`	`{÷(1+⍵*2)*0.5}`	
Inverse Hyperbolic Cosine	`{¯6○⍵}∆`	`{÷((⍵*2)-1)*0.5}`	
Inverse Hyperbolic Tangent	`{¯7○⍵}∆`	`{÷1-⍵*2}`	

The derivative rules for some scalar functions

Vector Functions

Vector functions take a vector as an argument and return a vector result. Their derivatives produce matrices. In the table below `f` and `g` are vector functions (that is, rank `1 1`) and `v` is a vector. The following definitions are assumed:

```
lm←{(⍳⍴⍵)∘.≥⍳⍴⍵}                    Lower mid array
alt←{⍵×(⍴⍵)⍴1 ¯1}                   Alternating sign
xp←{⍺×⍤(-(⍴⍴⍺)⌊⍴⍴⍵)⊢⍵}              Extended product
sh←{(⍺⌽⍳⍴⍴⍵)⍉⍵}                     Shift axes
```

Name	Definition	Equivalent	Note
Taylor expansion	`f v+dv`	`(f v)+(f ∆ v)+.×dv`	`dv←v×0.000001`
Sum	`(f+g)∆ v`	`(f ∆+g ∆)v`	
Difference	`(f-g)∆ v`	`(f ∆-g ∆)v`	
Product	`(f×g)∆ v`	`((f xp g ∆)+g xp f ∆)v`	
Quotient	`(f÷g)∆ v`	`(((f ∆)-(f÷g)xp g ∆)xp(÷g))v`	
Outer product	`(f∘.×g)∆ v`	`((f∘.×g ∆)+0 2 1⍉f ∆∘.×g)v`	
Composition	`(f g)∆ v`	`((f ∆ g)+.×g ∆)v`	
Inverse	`fi ∆ v`	`⌹(f ∆ fi)v`	`fi←f⍣¯1`
Matrix multiplication	`f ip g ∆ v`	`(f ip(g ∆)v)+(f ∆)ip g v`	
Reverse	`{⌽⍵}∆`	`{⌽id⍴⍵}`	
Transpose	`{⍉⍵}∆`	`{id⍴⍵}`	
Enclose	`{⊂⍵}∆`	`{⊂⍤1⊢id⍴⍵}`	
Plus-scan	`{+\⍵}∆`	`{lm ⍵}`	
Minus-scan	`{-\⍵}∆`	`{alt∘lm ⍵}`	
Times-scan	`{×\⍵}∆`	`{(×\⍵)×⍤¯1⊢(lm ⍵)÷⍤1⊢⍵}`	
Divide-scan	`{÷\⍵}∆`	`{(÷\⍵)×⍤¯1⊢(alt∘lm ⍵)÷⍤1⊢⍵}`	

The derivative rules for some vector functions

Note that the derivative of `{÷\⍵}` is expected to fail with a "Divide by zero" error if there is a `0` in the vector argument. Unless, of course, there is just one `0` and it is in the first position. Unfortunately, the expression for the derivative offered here, fails in this case when it should give a result.

```
      {÷\⍵}∆ 0 3 7
1        0 0
0.333333 0 0
2.33333  0 0
      {(÷\⍵)×⍤¯1⊢(alt∘lm ⍵)÷⍤1⊢⍵}0 3 7
DOMAIN ERROR: Divide by zero
```

Reductions

Reductions produce rank `0 1` functions. That is each vector within the argument's frame is reduced to a scalar. In effect, they return results with one fewer dimension than that of the argument – except for scalars which return their argument unchanged.

This means that the derivative of a reduction produces a result with the same shape as that of its argument.

Here are the derivatives of four commonly encountered reductions:

Name	Definition	Equivalent	Note
Plus-reduce	`{+/⍵}∆⍤1`	`{(⍴⍵)⍴1}⍤1`	
Minus-reduce	`{-/⍵}∆⍤1`	`{alt(⍴⍵)⍴1}⍤1`	
Times-reduce	`{×/⍵}∆⍤1`	`{×/⍵*⍤1⊢~id⍴⍵}⍤1` `{(×/⍵)÷⍵}⍤1`	
Divide-reduce	`{÷/⍵}∆⍤1`	`{(÷/⍵)÷alt ⍵}⍤1`	if ~0∊⍵

The derivatives of some reductions

Some care must be taken with max-reduce `{⌈/⍵}` and min-reduce `{⌊/⍵}`. This can be seen, as follows:

```
   x←5 3 2 5 4
   {⌈/⍵}∆above x
1 0 0 1 0
   {⌈/⍵}∆below x
0 0 0 0 0
   {⌈/⍵}∆ x
0.5 0 0 0.5 0
```

A similar difficulty arises for `{⌊/⍵}`. Of course, the reason stems from the fact that `⌈` and `⌊` are not differentiable functions everywhere; likewise `{×⍵}`, `{∨/⍵}` and `{∧/⍵}` and a number of others.

References

[0] *“The Derivative Operator”* K.E. Iverson, Proceedings of APL79: ACM 0-89791-005-2/79/0500-0347.

[1] *“The Derivative Revisited”* M. Powell, May 2020
https://aplwiki.com/wiki/File:1_The_Derivative_Revisited.pdf

[2] *“The Derivative Rules”* M. Powell, May 2020
https://aplwiki.com/wiki/File:2_The_Derivative_Rules.pdf

[3] "*Einstein Notation*", https://en.wikipedia.org/wiki/Einstein_notation

[4] *“Tensor Calculus”* J.L. Synge & A. Schild, Dover Publications, 1949.

[5] "*General Theory of Relativity*" P.A.M. Dirac, Princeton Landmarks in Physics, 1996

[6] "*Tensor Analysis, Theory & Applications*" I S Sokolnikoff, John Wiley & Sons, Inc. 1951