Excursions in Computing Science: Week 7c. Coordinates, Angles and Reality

T. H. Merrett*
McGill University, Montreal, Canada

June 10, 2015

I. Prefatory Notes

A. Reality

Gonna jump down, spin around, pick a bale of cotton.
Gonna jump down, spin around, pick a bale a day.
Norman Luboff, Harry Belafonte and William Attaway

1. Vectors are real.

   - Independent of coordinate axes, so
   - transform in a certain way when we change the axes.

Example transformations:

$$\text{rotate} \begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} c & s \\ -s & c \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix}$$

$$\text{reflect } x \begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} -1 & \\ & 1 \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix}$$

Note two assumptions underlying this Week: all the coordinate systems considered have common origin and common units.

So what are *not* vectors?

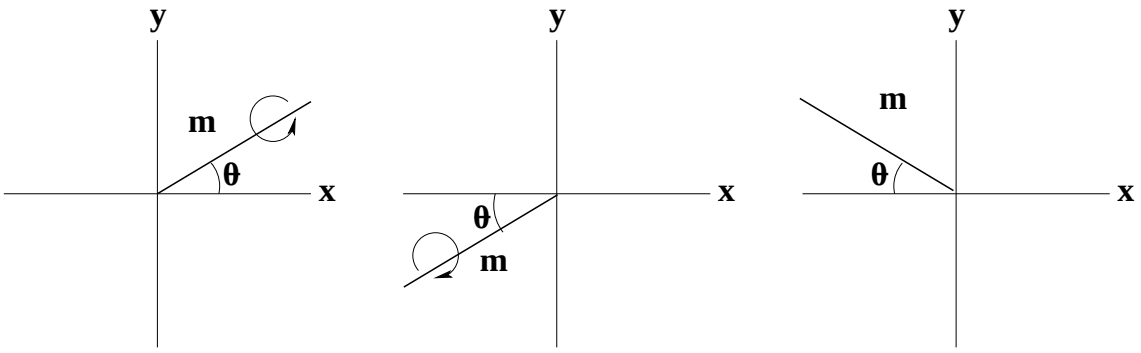
A twirl is not:

it has magnitude m and direction θ,

so $x = \cos\theta$ and $y = \sin\theta$

but it does not reflect the way a vector does.

 The author gratefully acknowledges support from the taxpayers of Québec and of Canada who have paid his salary and research grants while this work was developed at McGill University, and from his students and their funding agencies.

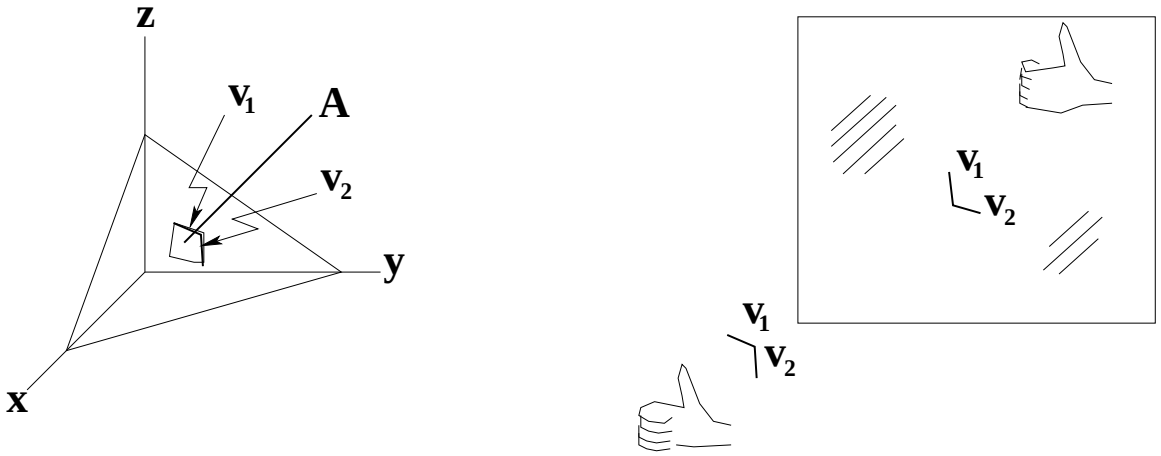


Twirl (m, θ) **Reflected x→ −x** **This is not the reflection**

We get $m' = -m$, i.e., $x' = -x$ and $y' = -y$

instead of $\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} -1 & \\ & 1 \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix}$

In 3D, an *area* is like a twirl: it can have an *orientation* to distinguish above from below.



We saw that a right-handed twirl becomes a left-handed twirl in the mirror.

Similarly the direction of turn needed to rotate v_1 into v_2 is reversed in the mirror. This direction can be taken to determine the orientation of the parallelopiped area defined by v_1 and v_2.

In some sense, $v_1 v_2 = -v_2 v_1$: the "product" is anticommutative. We'll follow up this essential insight shortly (Note 6).

2. Some pairs are not vectors: their components are not coordinates.

$$\begin{pmatrix} \text{apples}' \\ \text{oranges}' \end{pmatrix} \stackrel{??}{=} \begin{pmatrix} c & s \\ -s & c \end{pmatrix} \begin{pmatrix} \text{apples} \\ \text{oranges} \end{pmatrix}$$

This is not a totally hokey example. Information retrieval (I.R.) often uses "vectors" to capture the content of documents.

	around	bale	cotton	day	down	jump	pick	spin	
doc1(	1	1	1	0	1	1	1	1	)
doc2(	1	1	0	1	1	1	1	1	)

I.R. even uses dot products (Week 2, Note 5) to detect similarity between documents:

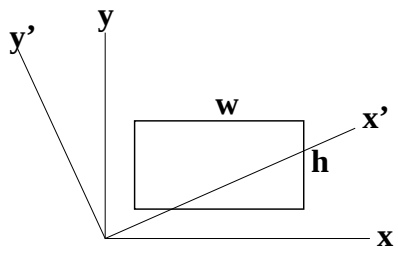
$$(\text{doc1} \ . \ \text{doc2})/(|\text{doc1}||\text{doc2}|) = 6/(\sqrt{7}\sqrt{7}).$$

But documents are not vectors: it is not meaningful to rotate or reflect the axes.

3. Even pairs of numbers from geometry, where rotating and reflecting *are* meaningful, are not always vectors. Let's try

$$\begin{pmatrix} \text{height} \\ \text{width} \end{pmatrix}$$

Here, no matter what the axes do, these numbers should not change.



What kind of thing remains invariant no matter what the axes do?

As with a vector, this thing, this pair of numbers, has a reality independent of the choice of coordinate axes. But the components of this one do not change if axes are rotated or reflected.

How about a matrix whose *eigenvalues* are w and h?

$$\begin{aligned} T\vec{v_1} &= w\vec{v_1} \\ T\vec{v_2} &= h\vec{v_2} \end{aligned}$$

For example, given the axes x and y shown,

$$\begin{aligned} T &= \begin{pmatrix} w & \\ & h \end{pmatrix} \\ v_1 &= \begin{pmatrix} 1 \\ 0 \end{pmatrix} \\ v_2 &= \begin{pmatrix} 0 \\ 1 \end{pmatrix} \end{aligned}$$

Then, for axes x' and y', related to x and y by rotation R,

$$\vec{v_1'} = R\vec{v_1} = \begin{pmatrix} c & s \\ -s & c \end{pmatrix} \begin{pmatrix} 1 \\ 0 \end{pmatrix}$$

and

$$RTR^{-1}\vec{v_1'} = RTR^{-1}R\vec{v_1} = RT\vec{v_1} = Rw\vec{v_1} = wR\vec{v_1} = w\vec{v_1'}$$

This suggests that T transforms to the new axes as $T' = RTR^{-1}$.

$$\begin{aligned} \text{Hence } T'\vec{v_1'} &= w\vec{v_1'} \\ \text{Similarly } T'\vec{v_2'} &= h\vec{v_2'} \end{aligned}$$

This is called a *tensor* transformation. Height and width (almost) form a "tensor". This tensor is a diagonal matrix, $\begin{pmatrix} w & \\ & h \end{pmatrix}$, when the axes are aligned with the rectangle, as x and y are.

This tensor is not diagonal for all coordinate axes, but we can see that it is a symmetric matrix.

$$T' = RTR^{-1} = \begin{pmatrix} c & s \\ -s & c \end{pmatrix} \begin{pmatrix} w & \\ & h \end{pmatrix} \begin{pmatrix} c & -s \\ s & c \end{pmatrix}$$

A symmetric matrix, T, equals its own transpose, $T = T^T$.

In general we may think of a tensor loosely as a matrix describing some real thing, as opposed to an operation or transformation.

$T' = RTR^{-1}$ is symmetric because the inverse of R is the transpose of R, $R^{-1} = R^T$, which is the case for rotations, reflections and other "orthogonal" transformations of coordinate axes.

4. Maybe twirl is a tensor too.

Try $S = \begin{pmatrix} a & b \\ c & d \end{pmatrix}$ and reflect in y by reversing the direction of x using the reFlection matrix F to give the tensor transformation FSF^{-1}

$$-\begin{pmatrix} a & b \\ c & d \end{pmatrix} = \begin{pmatrix} -1 & \\ & 1 \end{pmatrix} \begin{pmatrix} a & b \\ c & d \end{pmatrix} \begin{pmatrix} -1 & \\ & 1 \end{pmatrix} = \begin{pmatrix} a & -b \\ -c & d \end{pmatrix}$$

(Remember, Note 1 found out that the reflection just changes the sign of the twirl, i.e., of the tensor representing it.)

So $a = 0 = d$.

Any reflection will give a similar sign change, so let's see what reflecting in the line $x = y$ gives us: $F = \begin{pmatrix} & 1 \\ 1 & \end{pmatrix}$

$$-\begin{pmatrix} & b \\ c & \end{pmatrix} = \begin{pmatrix} & 1 \\ 1 & \end{pmatrix} \begin{pmatrix} & b \\ c & \end{pmatrix} \begin{pmatrix} & 1 \\ 1 & \end{pmatrix} = \begin{pmatrix} & c \\ b & \end{pmatrix}$$

and so $c = -b$.

Unfortunately, we've gone too far. We now have only one number, b, to describe a twirl, which we saw in Note 1 requires two numbers, m and θ.

So maybe two dimensions is too small to contain a twirl. This rather makes sense now that we think of it.

Let's see if we can decribe a twirl in three dimensions.

First note that $\begin{pmatrix} & -b \\ b & \end{pmatrix}$ is an *antisymmetric* matrix: it equals the negative of its transpose.

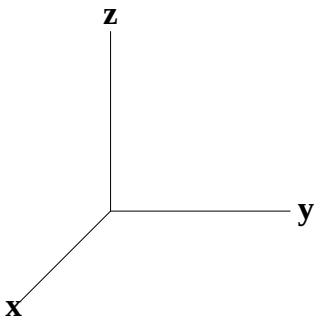
So we'll try an antisymmetric matrix in 3D. A 3×3 antisymmetric matrix has three components.

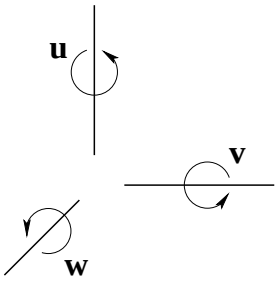
$$\begin{pmatrix} & u & v \\ -u & & w \\ -v & -w & \end{pmatrix}$$

Try reflecting in the yz plane: $x \leftrightarrow -x$

$$\begin{pmatrix} -1 & & \\ & 1 & \\ & & 1 \end{pmatrix} \begin{pmatrix} & u & v \\ -u & & w \\ -v & -w & \end{pmatrix} \begin{pmatrix} -1 & & \\ & 1 & \\ & & 1 \end{pmatrix} = \begin{pmatrix} & -u & -v \\ u & & w \\ v & -w & \end{pmatrix}$$

This almost just changes the sign of the matrix. Is it right?





Yes, if we interpret w as the x-component of the twirl, v as the y-component and u as the z-component. Check the diagram carefully!

Let's see what happens if we rotate in the xy plane.

$$\begin{pmatrix} c & s & \\ -s & c & \\ & & 1 \end{pmatrix} \begin{pmatrix} & u & v \\ -u & & w \\ -v & -w & \end{pmatrix} \begin{pmatrix} c & -s & \\ s & c & \\ & & 1 \end{pmatrix} = \begin{pmatrix} & u & sw+cv \\ -u & & cw-sv \\ -(sw+cv) & -(cw-sv) & \end{pmatrix}$$

This should be, and is, the same result we would get with $\begin{pmatrix} w \\ v \\ u \end{pmatrix}$ being just a vector, transformed in the usual vector way,

$$\begin{pmatrix} c & -s & \\ s & c & \\ & & 1 \end{pmatrix} \begin{pmatrix} w \\ v \\ u \end{pmatrix}$$

So a twirl, while transforming like a vector under rotation, is in general a tensor; for instance, it does not transform like a vector under reflection.

(Even though "twirl" is in one sense a rotation, we are here looking at it as a "real thing" so the matrix representing it is a tensor—as opposed to the quite different matrix that describes the *operator*, rotation.)

5. Twirl and area are "pseudovectors" or "axial vectors" in Willard Gibbs' vector analysis (which is widely used in spatial science). We now know that they are really tensors. It is just a coincidence that 3×3 antisymmetric tensors have 3 components, like a vector. This does not happen in two dimensions (1 component) or four dimensions (6 components).

Vector analysis generates pseudovectors by a "cross product" of two vectors: $A = v_1 \times v_2 = -v_2 \times v_1$, to use the area example from Note 1.

Vector analysis is unsatisfactory because

a) it is not a closed system: operating on vectors we get things that are not vectors (and, worse, they *look like* vectors);

b) it only works in three dimensions and does not generalize to more, or fewer, dimensions.

Can we make better abstractions for spatial entities, instead of vectors?

We need a formalism

- which is independent of coordinate axes;
- which captures the notion of area being the anticommutative combination of two vectors;
- which does not depend on the number of dimensions of the space.

B. Interval Algebra

6. Vectors and Areas and .. All Together

- Parts of space are lines, areas, volumes, ..
- We'll ignore absolute position and consider only direction and magnitude.
- We'll take the basis elements to be orthonormal and anticommutative.

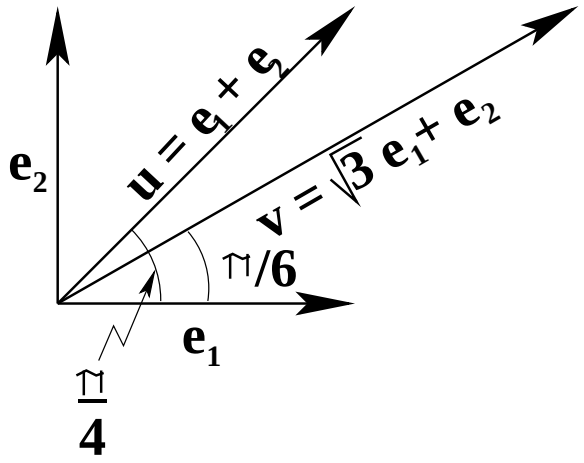
(We'll use the word "elements" instead of "vectors": some but not all elements can be thought of as vectors.)

1. The basis elements are e_1 and e_2, which are defined to have the following properties.

$$\begin{aligned} e_1 e_1 &\stackrel{\text{def}}{=} 1 \\ e_2 e_2 &\stackrel{\text{def}}{=} 1 \\ e_{12} &\stackrel{\text{def}}{=} e_1 e_2 \stackrel{\text{def}}{=} -e_2 e_1 \end{aligned}$$

2. An arbitrary element can be a linear combination of basis elements. Its product with itself is the square of its length or magnitude.

$$\begin{aligned} u &= e_1 + e_2 \\ uu &= (e_1 + e_2)(e_1 + e_2) = 1 + 1 = 2 \\ v &= \sqrt{3}e_1 + e_2 \\ vv &= (\sqrt{3}e_1 + e_2)(\sqrt{3}e_1 + e_2) = 3 + 1 = 2^2 \end{aligned}$$



3. The product of two different elements gives their magnitudes times the cosine and sine of the angle between them.

$$\begin{aligned} uv &= (e_1 + e_2)(\sqrt{3}e_1 + e_2) \\ &= \sqrt{3} + 1 + (1 - \sqrt{3})e_{12} \\ &= 2\sqrt{2}\left(\frac{\sqrt{3}+1}{2\sqrt{2}} + \frac{1-\sqrt{3}}{2\sqrt{2}}e_{12}\right) \\ &= 2\sqrt{2}(\cos(\pi/6 - \pi/4) + \sin(\pi/6 - \pi/4)e_{12}) \\ (ce_1 + se_2)(c'e_1 + s'e_2) &= (cc' + ss') + (cs' - c's)e_{12} \\ &= \cos(v - u) + \sin(v - u)e_{12} \end{aligned}$$

where $\cos(v-u)$ and $\sin(v-u)$ are respectively the cosine and sine of the angle from u to v: an *interval* from a to b is $b-a$ because adding s to it gives b.

7. Rotation

Let's have a magnitude operator ($\mid v \mid$ is an alternative notation),

$$\mathrm{mag}(v) = \mid v \mid = \sqrt{vv} = \text{length of } v$$

and a normalizing operator (${}^n v$ is an alternative notation),

$$\mathrm{norm}(v) = {}^n v = v/\mathrm{mag}(v) : \ \mathrm{norm}(v)\mathrm{norm}(v) = 1; \ v\,\mathrm{norm}(v) = \mathrm{mag}(v)$$

and $\mathrm{norm}(v)\mathrm{norm}(u)u = \ \mathrm{norm}(v)\mathrm{mag}(u)$, which rotates u into the direction of v.

$$\begin{aligned} \text{Try } \mathrm{norm}(u) &= ce_1 + se_2 \\ v &= \mathrm{mag}(v)(c'e_1 + s'e_2) = xe_1 + ye_2 \\ \mathrm{norm}(u)\mathrm{norm}(v) &= (cc' + ss') + (cs' - sc')e_{12} \\ &= C + Se_{12} \end{aligned}$$

where $C = \cos(v-u)$ and $S = \sin(v-u)$ as in Note 6. Compare this with 2-numbers, $C + iS$.

If we note that $e_{12}e_{12} = e_1e_2e_1e_2 = -e_1e_2e_2e_1 = -1$, we seem to find that e_{12} is the square root of -1. It's better to think of e_{12} as a $\pi/2$ rotation when postmultiplied (or a $-\pi/2$ rotation when premultiplied):

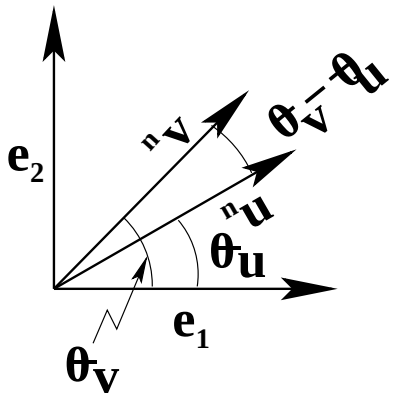
$$\begin{aligned} e_1e_{12} &= e_2 & \qquad e_{12}e_2 &= e_1 \\ e_2e_{12} &= -e_1 & \qquad e_{12}e_1 &= -e_2 \end{aligned}$$

(It is even better to think of e_{12} as a *plane*: see Note 11, below.)

So what is the meaning of $C + Se_{12}$?

$$\begin{aligned} u(C - Se_{12}) &= (xe_1 + ye_2)(C + Se_{12}) \\ &= (Cx - Sy)e_1 + (Sx + Cy)e_2 \\ &= (e_1\ e_2)\begin{pmatrix} C & -S \\ S & C \end{pmatrix}\begin{pmatrix} x \\ y \end{pmatrix} \end{aligned}$$

It's the rotation that rotates u onto v (the figure uses ${}^n v$ for $\mathrm{norm}(v)$): $u\mathrm{norm}(u)\mathrm{norm}(v) = v = \mathrm{norm}(v)\mathrm{norm}(u)u$.

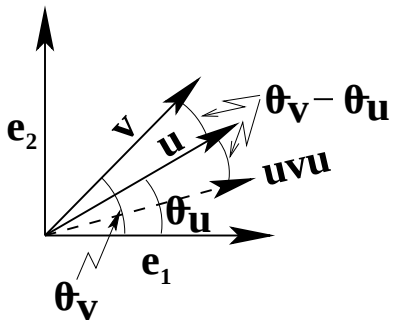


8. Reflection

If uuv and vuu rotate $u \to v$ what is uvu?

Let's try it with u and v normalized.

$$\begin{aligned} u &= c'e_1 + s'e_2 \\ v &= ce_1 + se_2 \\ uvu &= (c'e_1 + s'e_2)(ce_1 + se_2)(c'e_1 + s'e_2) \\ &= Ce_1 + Se_2 \text{ where} \\ C &= \cos(\theta_u - \theta_v + \theta_u) = \cos(\theta_u - (\theta_v - \theta_u)) \\ S &= \sin(\theta_u - \theta_v + \theta_u) = \sin(\theta_u - (\theta_v - \theta_u)) \end{aligned}$$



uvu is the *reflection* of v in u.

(Another viewpoint: since $w(vu)$ rotates w by the angle between v and u, so $u(vu)$ is the reflection of v in u.)

Note that the *projection* of v in u is $(uvu + v)/2$, which can be written as a relationship among the reflection operator, F, the identity operator, I, and the projection operator, P: $P = (F + I)/2$.

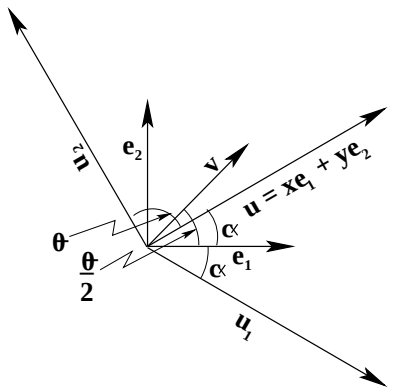
Note finally that a rotation is two reflections:

1. in e_1;
2. in "half-u", an element whose angle with e_1 is half the angle we wish to rotate through.

(We'll use the subscript J to indicate half-angles, since J sort of looks like 2 upside-down.)

$$\begin{aligned} v &= xe_1 + ye_2 & c = \cos\theta \quad c_J = \cos\theta/2 \\ u_J &= c_J e_1 + s_J e_2 & s = \sin\theta \quad s_J = \sin\theta/2 \\ u_J e_1 v e_1 u_J &= (c_J - s_J e_{12}) v (c_J + s_J e_{12}) \\ &= (e_1\ e_2) \begin{pmatrix} c & -s \\ s & c \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} \end{aligned}$$

which is the rotation. (Recall that $c = c_J^2 - s_J^2$ and $s = 2c_J s_J$.) $\theta/2 + \theta/2 + \alpha - \alpha = \theta$:



9. 3D rotations

Outside of a 2-D plane we can't use $C + Se_{12}$ in 3-D:

$$e_3(C + Se_{12}) = Ce_3 + Se_{123}$$

(Note the extension of the rule for combining basis elements:

$$e_3e_{12} = e_3e_1e_2 = -e_1e_3e_2 = e_1e_2e_3 \stackrel{\text{def}}{=} e_{123})$$

So let's try two reflections:

$$\begin{array}{rcl} \text{rotate } v & = & xe_1 + ye_2 + ze_3 \\ \text{in plane } P & = & re_{12} + pe_{23} + qe_{31} \end{array}$$

with P normalized: $p^2 + q^2 + r^2 = 1$.

$(c_J - s_JP)v(c_J + s_JP) =$

$$(e_1\ e_2\ e_3)\left(\left(\begin{array}{ccc} c & -sr & sq \\ sr & c & -sp \\ -sq & sp & c \end{array}\right) + (1-c)\left(\begin{array}{c} p \\ q \\ r \end{array}\right)(p,q,r)\right)\left(\begin{array}{c} x \\ y \\ z \end{array}\right)$$

Note that $pe_1 + qe_2 + re_3 \perp P = re_{12} + pe_{23} + qe_{31}$.

Note also that $\left(\begin{array}{c} p \\ q \\ r \end{array}\right)$ is an eigenvector of the rotation matrix: what is the significance of that?

Now *two* rotations:

$$\begin{array}{rl} \text{by } (c,s) \text{ about} & pe_1 + qe_2 + re3 \\ \text{then by } (c',s') \text{ about} & p'e_1 + q'e_2 + r'e3 \\ \Downarrow & \\ \text{a rotation by } (c'',s'') \text{ about} & p''e_1 + q''e_2 + r''e3 \end{array}$$

$$(c_J + s_J(re_{12} + pe_{23} + qe_{31}))(c'_J + s'_J(r'e_{12} + p'e_{23} + q'e_{31}))$$
$$= c''_J + s''_J(r''e_{12} + p''e_{23} + q''e_{31})$$

where

$$\begin{array}{rcl} c''_J & = & c_Jc'_J - s_Js'_J(rr' + pp' + qq') \\ s''_Jr'' & = & s_Jc'_Jr + c_Js'_Jr' + s_Js'_J(qp' - pq') \\ s''_Jp'' & = & s_Jc'_Jp + c_Js'_Jp' + s_Js'_J(rq' - qr') \\ s''_Jq'' & = & s_Jc'_Jq + c_Js'_Jq' + s_Js'_J(pr' - rp') \end{array}$$

Note that in 3-D all the angles are half angles.

Note that 3-D rotations do not commute.

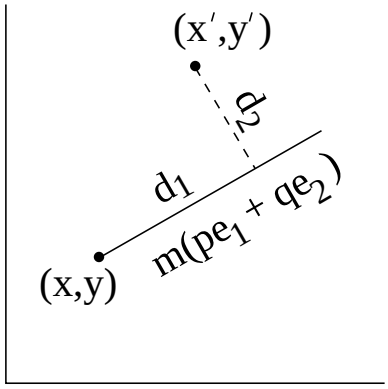
10. Intervals plus locations. The intervals described by the interval algebra have magnitude and orientation but no location.

Thus they cannot solve problems such as finding the distance from a point to a line.

We must work with both points and line intervals.

For a start, the line interval must be anchored to a point, say (x, y).

Then we can formulate the problem as "find the distance from a point (x', y') to the line that is the interval $m(pe_1 + qe_2)$ starting from point (x, y).



Here, m is the magnitude of the interval and p and q give its orientation (normalized so $p^2+q^2 = 1$). (x, y) and (x', y') are two points, which are beyond the scope of the interval algebra, and our task is to find the length of the dashed line, which is the distance from (x', y') to the interval starting at (x, y), and to ascertain that this vertical actually meets the original line within the interval.

We can find the dashed line as an interval, $p'e_1+q'e_2$, which we might as well normalize, $p'^2+q'^2 = 1$. This does not locate the interval, but we can do that by making it start at (x', y'). For orthogonality, either $pp' + qq' = 0$ or, directly, take the interval product with the containing plane

$$p'e_1 + q'e_2 = (pe_1 + qe_2)e12 = -qe_1 + pe_2$$

We consider as unknowns the two distances to the intersection of the solid and dashed lines, d_1 from (x, y), and d_2 from (x', y').

Then we can switch to vector notation and write

$$\begin{pmatrix} x \\ y \end{pmatrix} + d_1 \begin{pmatrix} p \\ q \end{pmatrix} = \begin{pmatrix} x' \\ y' \end{pmatrix} - d_2 \begin{pmatrix} -q \\ p \end{pmatrix}$$

(Why is there a $-$ sign before d_2?)

This becomes

$$\begin{pmatrix} p & -q \\ q & p \end{pmatrix} \begin{pmatrix} d_1 \\ d_2 \end{pmatrix} = \begin{pmatrix} x' - x \\ y' - y \end{pmatrix}$$

which is easily solved, especially since the determinant is 1.

If $d_1 \leq m$, the dashed line does meet the solid line within the given interval.

11. Interval algebra in 3D. We can get a better feeling for interpreting the interval algebra from using it in 3D than from using it in 2D.

First, let's define the *order* of an interval, or part of an interval, as the number of subscripts on the basis elements. Intervals can be of homogenous order, such as the line interval $pe_1 + qe_2 + re_3$

(order 1) or the plane interval $pe_{23} + qe_{31} + re_{12}$ (order 2), or of mixed order, such as the product $pp' + qq' + rr' + (pq' - qp')e_{23} + (qr' - rq')e_{31} + (rp' - pr')e_{12}$.

In three dimensions, order-1 intervals are line intervals and order-2 intervals are plane intervals. The magnitude of a line interval is its length.

A plane interval can be considered equally as a plane area (the "area" is its magnitude) or as a twirl. In either case. it can also be thought of, in 3D, as the line interval orthogonal to it ($pe_{23}+qe_{31}+re_{12}$ or $pe_1 + qe_2 + re_3$), and in the twirl case, this line interval is the axis of rotation.

For those familiar with Gibbs' vector notation, this orthogonal to a plane interval is the "pseudovector" that is the "cross product" of the two vectors in the plane whose magnitudes times the sine of the angle between them is the magnitude of the cross product. It is much better to think of plane intervals than pseudovectors. For one thing, intervals generalize to any number of dimensions. Pseudovectors do not.

It is also useful to be able to extract components of any given order, so we define a **cmpt** operator with a parameter specifying the order of the component to be extracted.

$$\mathbf{cmpt}(0, (pe_1 + qe_2 + re_3)(p'e_1 + q'e_2 + r'e_3) = pp' + qq' + rr'$$

and similarly for $\mathbf{cmpt}(2, (pe_1 + qe_2 + re_3)(p'e_1 + q'e_2 + r'e_3))$, etc.

The Gibbs' "dot product" between the two vectors is given by $\mathbf{cmpt}(0,)$ and the "cross product" is given by $\mathbf{cmpt}(2,)$.

Now, the interval product between two homogeneous intervals of the same order gives the angle between them and the plane common to them.

$$uv = \text{mag}(u)\text{mag}(v)(c + s\,\text{normPlane}(u, v))$$

For example,

$$\begin{aligned}(pe_1 + qe_2 + re_3)(p'e_1 + q'e_2 + r'e_3) &= pp' + qq' + rr' + (pq' - qp')e_{23} + (qr' - rq')e_{31} + (rp' - pr')e_{12} \\ &= -(pe_{23} + qe_{31} + re_{12})(p'e_{23} + q'e_{31} + r'e_{12})\end{aligned}$$

where

$$\begin{aligned}\text{mag}(u) &= 1 = \text{mag}(v) \\ c &= pp' + qq' + rr' \\ s &= \sqrt{(pq' - qp')^2 + (qr' - rq')^2 + (rp' - pr')^2}\end{aligned}$$

and normPlane(u, v) is the normalized plane consisting of the second-order component divided by s.

The interval product between a homogeneous interval and a containing interval gives the interval within the containing interval that is orthogonal to the first interval. Here is a slightly more general example.

$$\begin{aligned}(p'e_1 + q'e_2 + r'e_3)(pe_1 + qe_2 + re_3) &= (pp' + qq' + rr')e_{123} + (pq' - qp')e_3 + (qr' - rq')e_1 + (rp' - pr')e_2 \\ &= (p'e_{23} + q'e_{31} + r'e_{12})(pe_1 + qe_2 + re_3)\end{aligned}$$

where, if the line (order-1) intervals are contained, respectively, in the plane (order-2) intervals, $pp' + qq' + rr' = 0$ because the line intervals are orthogonal, respectively, to the line intervals orthogonal ("normals") to the plane intervals. (The word "normals" can be confused with the word "normal", describing an interval of magnitude 1, so we do not continue to use it.)

We can use these interpretations to find a new set of orthonormal axes (orthogonal to each other and normalized) given one desired axis. We work a specific example in which $f_1 = (e_1+e_2+e_3)/\sqrt{3}$.

First, find the plane interval orthogonal to f_1

$$(e_1 + e_2 + e_3)e_{123}/\sqrt{3} = (e_{23} + e_{31} + e_{12})/\sqrt{3}$$

Second, find any line interval in this plane: the condition is that $p'e_1 + q'e_2 + r'e_3$ is orthogonal to f_1 so $p' + q' + r' = 0$. This eliminates one of the three unknowns, and we might as well make a choice among the others which is as simple as possible. So suppose $r' = 0$ and $q' = -p'$:

$$f_2 = (e_1 - e_2)/\sqrt{2}$$

Third, find a second line interval in the orthogonal plane which is orthogonal to the first.

$$\begin{aligned} f_3 &= (e_1 - e_2)(e_{23} + e_{31} + e_{12})/\sqrt{3}/\sqrt{2} \\ &= (e_1 + e_2 - 2e_3)/\sqrt{6} \end{aligned}$$

Check that $f_j f_j = 1$ and $f_j f_k = -f_k f_j$ if $j \neq k$, the same properties that the e_j have..

Finally, observe that the matrix transforming from the e_j to the f_k is just given by the coefficients.

$$\begin{pmatrix} f_1 \\ f_2 \\ f_3 \end{pmatrix} = \begin{pmatrix} 1/\sqrt{3} & 1/\sqrt{3} & 1/\sqrt{3} \\ 1/\sqrt{2} & -1/\sqrt{2} & \\ 1/\sqrt{6} & 1/\sqrt{6} & -2/\sqrt{6} \end{pmatrix} \begin{pmatrix} e_1 \\ e_2 \\ e_3 \end{pmatrix}$$

Check that the inverse of this matrix is its transpose, and convince yourself that the transformation of *coordinates*, if the *space* were to be rotated the same way, relative to the original axes, e_j, is this transpose.

12. Summary

(These notes show the trees. Try to see the forest!)

- Vectors are real things, independent of coordinates.
- So where they are written in terms of coordinates, these coordinates must transform correctly under rotation, reflection, projection and inversion: $X\vec{v}$.
- *Some* real things are not vectors, but tensors, and so tensor elements must also transform correctly: XTX^{-1}.
- Clifford or geometric or angle or interval algebra:
  - parts of space: lines, areas, volumes, ..;
  - ignore position, consider only magnitude, direction;
  - basic elements are orthonormal and commutative.
- 2-D rotation from u to v is uuv or vuu.
- Reflection of v in u is uvu.
- 3-D rotation by (c, s) about $re_{12} + pe_{23} + qe_{31}$ ..
- Two 3-D rotations need half angles and are not commutative.
- Intervals have no locations, only magnitudes and orientations, so the interval algebra must be supplemented by points if, say, distances are to be found.
- Interval products have a number of useful interpretations, including angles between lines and planes, and orthogonals to lines and planes.

NB In 2-D: $1, e_1, e_2, e_3, e_{12}$. In 3-D: $1, e_1, e_2, e_3, e_{23}, e_{31}, e_{12}, e_{123}$.

13. Appendix: Summary of vector and matrix operations

+

$$\begin{aligned}\vec{u}+\vec{v} &= \begin{pmatrix} u_1+v_1 \\ u_2+v_2 \end{pmatrix} \\ A+B &= \begin{pmatrix} a_{11}+b_{11} & a_{12}+b_{12} \\ a_{21}+b_{21} & a_{22}+b_{22} \end{pmatrix}\end{aligned}$$

•

$$\begin{aligned}\vec{u}.\vec{v} &= (u_1\ u_2)\begin{pmatrix} v_1 \\ v_2 \end{pmatrix} \\ &= u_1v_1+u_2v_2 \\ &= \mid \vec{u} \mid\mid \vec{v} \mid \cos(\angle(\vec{u},\vec{v})) \\ A\vec{u} &= \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix}\begin{pmatrix} u_1 \\ u_2 \end{pmatrix} \\ &= \begin{pmatrix} a_{11}u_1+a_{12}u_2 \\ a_{21}u_1+a_{22}u_2 \end{pmatrix} \\ \vec{u}A &= (u_1\ u_2)\begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix} \\ &= (u_1a_{11}+u_2a_{21}\ \ u_1a_{12}+u_2a_{22}) \\ AB &= \begin{pmatrix} a_{11}b_{11}+a_{12}b_{21} & a_{11}b_{12}+a_{12}b_{22} \\ a_{21}b_{11}+a_{22}b_{21} & a_{21}b_{12}+a_{22}b_{22} \end{pmatrix}\end{aligned}$$

$\otimes$

$$A\otimes B = \begin{pmatrix} a_{11}B & a_{12}B \\ a_{21}B & a_{22}B \end{pmatrix}$$

Clifford algebra

$$\begin{aligned}uv &= (u_1e_1+u_2e_2)(v_1e_1+v_2e_2) \\ &= u_1v_1+u_2v_2+(u_1v_2-u_2v_1)e_{12} \\ &= \vec{u}.\vec{v}+\mid \vec{u}\times\vec{v} \mid e_{12} \\ &= \mathrm{mag}(u)\mathrm{mag}(v)(\cos(\angle(\vec{u},\vec{v}))+\sin(\angle(\vec{u},\vec{v}))e_{12})\end{aligned}$$

(The third line does not use the Clifford algebra mag() operator because it is not Clifford algebra. It is a digression for those familiar with Gibbs' vector algebra.)

Compare $\begin{pmatrix} u_1 \\ u_2 \end{pmatrix}(v_1,v_2) = \begin{pmatrix} u_1v_1 & u_1v_2 \\ u_2v_1 & u_2v_2 \end{pmatrix}$

Finally, compare these with 2-numbers (Week 4: we use 2-number notation for the magnitude instead of the Clifford algebra mag() operator):

$$\begin{aligned}u+v &= u_1+v_1+i(u_2+v_2) \\ uv &= (u_1+iu_2)(v_1+iv_2) \\ &= u_1v_1-u_2v_2+i(u_1v_2+u_2v_1) \\ &= \mid u \mid e^{i\angle u} \mid v \mid e^{i\angle v} \\ &= \mid u \mid\mid v \mid e^{i(\angle u+\angle v)}\end{aligned}$$

II. The Excursions

You've seen lots of ideas. Now *do* something with them!

1. **Dot product.**
   a) The dot product (see Week 2 Note 5) of two normalized vectors in any number of dimensions equals the cosine of the angle between the vectors. Show this: i) use $(Xu)^T Xv = u^T v$ to discover that the dot product is invariant under any axis transformation, X, whose transpose is its inverse; and ii) use this invariance to reduce any two d-dimensional vectors, $\vec{u}$ and $\vec{v}$, to the two dimensions of their common plane.
   b) What is the angle between doc1 and doc2 in Note 2?
   c) How does the dot product of a vector, v, with itself relate to the interval algebra product vv in Note 6?
   d) How would you define the dot product of the interval algebra basis elements e_1 and e_2 so that for any vectors $u = u_1 e_1 + u_2 e_2$ and $v = v_1 e_+ v_2 e_2$, $u.v = u_1 v_1 + u_2 v_2$? Relate this to the definition of the dot product (Week 2 Note 5) in terms of some particular coordinate system. Why is the product $a_x b_x + a_y b_y$ of any coordinates a_x, a_y, b_x and b_y invariant, i.e., has the same value no matter what axes are used to specify the coordinates?
   e) Show that $\vec{u}.\vec{v}/\mid v \mid$ is the component of $\vec{u}$ along the direction of $\vec{v}$ for any two vectors $\vec{u}$ and $\vec{v}$.

2. Calculate the reflections in the yz plane of twirls pointing along each of the x, y and z axes, and explain why what you get is right.

3. Confirm that w, u and v in the 3D twirl tensor must refer to the x, y and z components, respectively.

4. Is there a way to use 2-numbers to represent 3D twirl as a 2×2 tensor?

5. Show that postmultiplyimg by e_{12} is the same as premultiplying by e_{21} in Note 7. What does this imply for expressing the rotation from u to v as a premultiplication?

6. What is the matrix for the reflection of $v = xe_1 + ye_2$ in $u = ce_1 + se_2$ (c and s are cosine and sine, respectively, so u is normalized)?

7. Why is $u(vu)$ the reflection of v in u? Explain in terms of the rotation, (uv). (Take u and v to be normalized.)

8. A ball moving along trajectory b bounces off a wall w. What is its new trajectory?

9. Explain why the projection of v on u is $(uvu + v)/2$. For $u = c'e_1 + s'e_2$ and $v = ce_1 + se_2$, give the matrices F (reflection) and P (projection). What is the significance of $P - I$, where I is the identity matrix?

10. Show that 3D rotation by angle (c, s) about $re_{12} + pe_{23} + qe_{31}$ is the matrix given in Note 9. Show that $(p, q, r)^T$ is an eigenvector (Note 1 of Week 8), find the corresponding eigenvalue, and explain what these mean.

11. Check the derivation of the expression for double rotation in 3D. How would we find p'', q'' and r''?

12. Compare rotating by $\pi/2$ about (1,0,0) then $\pi/2$ about (0,1,0) with rotating $\pi/2$ about (0,1,0) then $\pi/2$ about (1,0,0). Use both interval algebra and your hands and some physical object such as a book.

13. Using rotations (and other operations) in the interval algebra and a starting edge, e_1, find the other two edges of an equilateral triangle. How would this help you draw it with a graphics program?
    Once you've found the second edge, there are at least three ways of finding the third: figure them all out and compare.

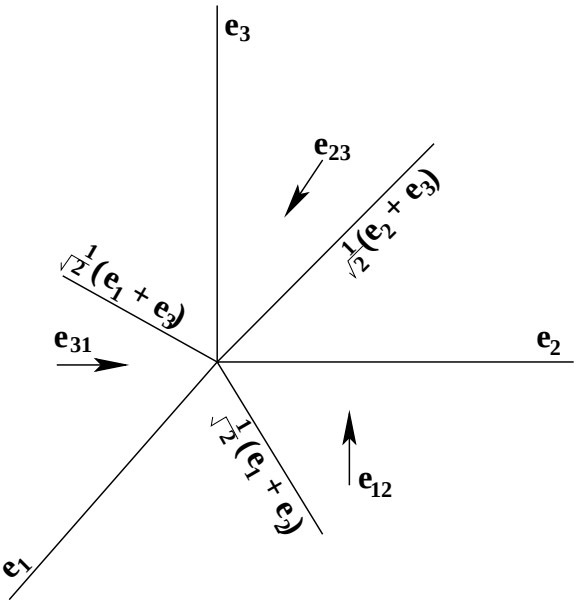
14. Rotate the equilateral triangle of the previous Excursion just enough to map it onto itself and show that the edges you found there do indeed map onto each other.

15. Why can the Interval Algebra not be used to find the intersection of two lines?

16. **Tetrahedron.** Using rotations (and other operations) in the interval algebra and the equilateral triangle of the previous Excursion, calculate the three edges needed to build it into an equilateral tetrahedron. How would you find the angles between the planes in the tetrahedron?

17. What is the 3-by-3 matrix that gives a 1/3 rotation (i.e., by $2\pi/3$) about the axis (1,1,1)? Check that this make sense: multiply it by itself once, then once more.

18. a)
    (Warmup and check.) What is the plane formed by the edges $\mathbf{e}_1$ and $(\mathbf{e}_2 + \mathbf{e}_3)/\sqrt{2}$? What is the angle between these two edges? What angle does the plane make with $\mathbf{e}_{12}$? (Keep all edges and planes normalized! Be careful about signs, and check what they mean!)
    b) Answer the questions from (a) for the edges $(\mathbf{e}_1 + \mathbf{e}_3)/\sqrt{2}$ and $(\mathbf{e}_2 + \mathbf{e}_3)/\sqrt{2}$.



    c) Examine and test the MATLAB function

```
% function [cos12,sin12,face12] = product(edge1,edge2)
% THM 070410    in file: product.m
% edge1: normalized 3-vector, e.g. [p1,q1,r1]
% edge2: normalized 3-vector, e.g. [p2,q2,r2]
% cos12 = p1p2+q1q2+r1r2
% sin12 = +sqrt(1-cos^2)
% face12: normalized 3-vector,
%  [(q1r2-r1q2)/sin12,(r1p2-p1r2)/sin12,(p1q2-q1p2)/sin12]
% (Works for planes as input, but use -cos12, -sin12)
```

```
function [cos12,sin12,face12] = product(edge1,edge2)
p1 = edge1(1); q1 = edge1(2); r1 = edge1(3);
p2 = edge2(1); q2 = edge2(2); r2 = edge2(3);
cos12 = p1*p2+q1*q2+r1*r2;
sin12 = sqrt(1-cos12^2):  % when might this be 0?
if abs(sin12)<10^-8 face12 = [0,0,0]; else
face12 = [(q1*r2-r1*q2)/sin12,(r1*p2-p1*r2)/sin12,(p1*q2-q1*p2)/sin12];
end
```

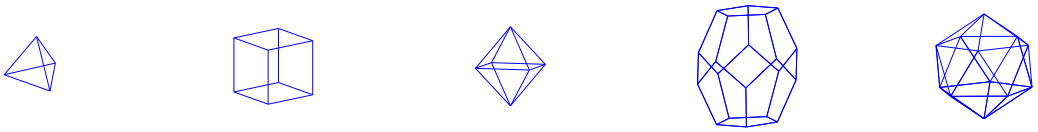
Why must we change the sign if `edge1` and `edge2` represent faces rather than edges on input? (Hint. Multiplying by $\mathbf{e}_{12}$ in 2D gives a quarter-rotation. Does multiplying by $\mathbf{e}_{123}$ in 3D also do this? What does a "quarter rotation" mean in this case for an edge? For a face? What is $\mathbf{e}_{123}\mathbf{e}_{123}$?)
d) (Warmup and check.) Rotate the edges $\mathbf{e}_1$ and $(\mathbf{e}_2 + \mathbf{e}_3)/\sqrt{2}$ through the angle you found in (a) so as to put them both in $\mathbf{e}_{12}$: this should give $\mathbf{e}_1$ itself and $\mathbf{e}_2$, respectively.
e) Rotate the edges from (b) so as to put them both in $\mathbf{e}_{12}$. Check that they have the same angle with each other that they did before rotating.
f) Find two additional normalized edges that share with each of the new edges from (e) the same angle you found in (b) that they have with each other. (Note that the solution is direct if the input edges are in $\mathbf{e}_{12}$ but would require iteration if the $\mathbf{e}_3$ components of the edges are nonzero: try it!)
g) Write a MATLAB function, `e12equiAngle()`, for (f), i.e., which given two edges in $\mathbf{e}_{12}$ finds an edge sharing with those two edges the angle that is between the input edges.
Write a MATLAB function, `equiAngle()`, which given *any* two edges finds an edge sharing with those two edges the angle that is between the input edges: find the plane of the given edges, rotate it into the $\mathbf{e}_{12}$ plane, use `e12equiAngle()` to find the new edge, and rotate this back again. (The next excursion gives a possible `rotate3D()` function interface.)
h) Rotate the edge from (f) that has the negative $\mathbf{e}_3$ component inversely to the rotation in (e). What is the resulting combination of this edge and the two original edges in (b)?

19. Inspect and run the following MATLAB function.

```
% function [pentcoords,pentedges,pentface] = pentagon(startcoords,startedge,pentface)
% THM 070409    in file: pentagon.m
% Makes pentagon of unit edges, given 3D coords for 1 vertex, 1 edge, 1 plane
% startcoords 3-vector, e.g. [0,0,0]
% startedge 3-vector, e.g. [1,0,0]
% pentface 3-vector, e.g.[0,0,1]      The plane in which the pentagon is made
% pentcoords 5*3 array, e.g. [0,0,0;1,0,0;..]
% pentedges 5*3 array, e.g. [1,0,0;..]
% uses rotate3D
function [pentcoords,pentedges,pentface] = pentagon(startcoords,startedge,pentface)
angle = 2*pi/5;
edgesIN = startedge';
planesIN = pentface';
pentedges = edgesIN;
pentcoords = startcoords';
[edgesOUT,planesOUT] = rotate3D(pentface,angle,edgesIN,planesIN);
for k = 1:4
   pentedges = [pentedges,edgesOUT];
   coords = pentcoords(k,:) + pentedges(k,:)
```

```
    pentcoords = [pentcoords,coords];
    [edgesOUT,planesOUT] = rotate3D(pentface,angle,edgesOUT,planesIN);
end
```

Write the function `rotate3D(plane,angle,edgesIN,planesIN)`, which rotates arbitrary sets of `edgesIN` and `planesIN` about `angle` in `plane`.
Write a program which calls `pentagon()` and uses `quiver3` to draw the resulting pentagon.

20. 

Above are the five "Platonic solids": the tetrahedron (4 faces), the cube (hexahedron, 6 faces), octahedron (8 faces), dodecahedron (12 faces) and icosahedron (20 faces). Use the techniques of the previous excursions to build them in MATLAB.
(The cube and octahedron do not need interval algebra machinery and their edges can be written down straight from pairs of coordinates. They make a good place to start. The tetrahedron can also be written down directly from coordinates, or it can be made from an equilateral triangle and an additional vertex out of the plane and equidistant from each vertex of the triangle; but it is good exercise to use interval algebra for this, following the **Tetrahedron** Excursion, above, or the notes on Clifford Algebra available from the course home page.)
By finding a way to draw the octahedron inside the cube and the icosahedron inside the dodecahedron, show that these are two pairs of "duals"—the faces of one of each pair correspond to the vertices of the other, and vice-versa. What is the dual of the tetrahedron?

21. Use the pdf notes "Clifford Algebra" for this week to find the coordinates of the centre of a tetrahedron (the point equidistant from each vertex) and to show that the angle between any two edges connecting the centre with two vertices is about $109^o 27'$.

22. How many colours are needed to colour the vertices of each of the Platonic solids, if no two vertices of the same colour may be joined by an edge? How many colours for the faces, if no two faces separated by an edge as a boundary may have the same colour? What about colouring vertices of polygons in 2D?

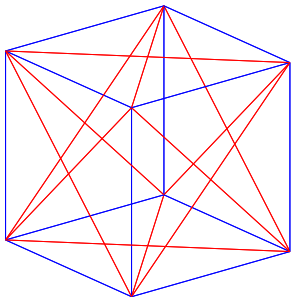
23. Confirm that the Platonic solids satisfy

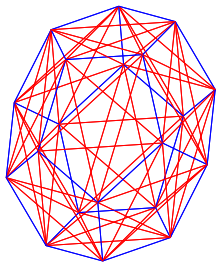
$$2 + E = F + V$$

where E is the number of edges, F is the number of faces and V is the number of vertices. Does this hold for any other figure?

24. How many spheres can be packed around a sphere of the same radius? (Hint: start with 2D and show that six circles pack a centre circle. What angle does each circle subtend at the centre? Approximately what proportion of the spherical surface area, $4\pi r$, is inside one of the packing spheres centred at distance r? Must the centres of the packing spheres form the vertices of one of the Platonic solids?)

25.





The red additions to the cube and the dodecahedron above are the paths of length 2. That is, since the cube has a blue edge (0,0,0)–(1,0,0) and a blue edge (1,0,0)–(1,1,0), then (0,0,0)–(1,1,0) will be a red edge.
Here are all the coordinate pairs for the cube, in two different orders: the set on the left is sorted by columns 4, 5 and 6; the set on the right is sorted by columns 1, 2 and 3.

0	0	1	0	0	0	0	0	0	0	0	1
0	1	0	0	0	0	0	0	0	0	1	0
1	0	0	0	0	0	0	0	0	1	0	0
0	0	0	0	0	1	0	0	1	0	0	0
0	1	1	0	0	1	0	0	1	0	1	1
1	0	1	0	0	1	0	0	1	1	0	1
0	0	0	0	1	0	0	1	0	0	0	0
0	1	1	0	1	0	0	1	0	0	1	1
1	1	0	0	1	0	0	1	0	1	1	0
0	0	1	0	1	1	0	1	1	0	0	1
0	1	0	0	1	1	0	1	1	0	1	0
1	1	1	0	1	1	0	1	1	1	1	1
0	0	0	1	0	0	1	0	0	0	0	0
1	0	1	1	0	0	1	0	0	1	0	1
1	1	0	1	0	0	1	0	0	1	1	0
0	0	1	1	0	1	1	0	1	0	0	1
1	0	0	1	0	1	1	0	1	1	0	0
1	1	1	1	0	1	1	0	1	1	1	1
0	1	0	1	1	0	1	1	0	0	1	0
1	0	0	1	1	0	1	1	0	1	0	0
1	1	1	1	1	0	1	1	0	1	1	1
0	1	1	1	1	1	1	1	1	0	1	1
1	0	1	1	1	1	1	1	1	1	0	1
1	1	0	1	1	1	1	1	1	1	1	0

a) Confirm that these coordinate pairs link up so as to give the red edges shown with the cube.
b) Examine the following MATLAB code which will make the links you checked in (a). It implements a simplified *natural composition* operator of the *relational algebra*. It is built in terms of three other relational algebra operators, *natural join, projection* and a family of operators that treat relations as sets of rows and produce set difference (−), union (`u`), intersection (`n`) and symmetric difference (+). (Note that this last operator is here applied to the set of *columns* of the relations being put together.)
Look up [Mer99, Database programming], implement these operators, and show that

`relationCompos()` applied to the coordinate pairs for the cube produces the red figures shown.
c) Run your `relationCompos()` on the coordinate pairs you got for the dodecahedron in an earlier excursion.

```
% function joinOut = relationCompos(joinIndices,joinIn1,joinIn2)
% THM 070420 in file relationCompos.m
% joinIndices 2*m array giving indices to be joined
% joinIn1 n1*m1 array
% joinIn2 n2*m2 array
% joinOut n*(m1-m+m2) array
% joinOut rows will be unduplicated if joinIn1 and joinIn2 rows are
% Uses relationSetOp(), relationJoin(), relationProject()
function joinOut = relationCompos(joinIndices,joinIn1,joinIn2)
sizIn1 = size(joinIn1);
sizIn2 = size(joinIn2);
sizInd = size(joinIndices(1,:));
%all = zeros(sizInd);               % indices for compareRows(): all columns
for k = 1:sizIn1(2) - sizInd(2) + sizIn2(2) all(k) = k; end
projIndices = relationSetOp('-',all',joinIndices(1,:)')
joinOut = relationProject(projIndices',relationJoin(joinIndices,joinIn1,joinIn2));
```

26. Combine the methods of Notes 10 and 11 to find the distance in three dimensions between a given point and a line made up of an interval and another given point as start point. (You will need to solve for three distances, and combine two of them to give the desired distance.)

27. Note 11 gives the expression $c + s$ normPlane(u, v).
a) Show that $c^2 + s^2 = 1$ so that it is plausible to interpret c and s as cos() and sin(), respectively. (Try it in two dimensions first.)
b) Use the axis-rotating method of Note 11 to change axes so that a new f_3 is orthogonal to the plane containing u and v, and thereby establish that c and s really are cos() and sin() in the two-dimensional f_{12} plane.

28. Show that $pp' + qq' + rr' = 0$ also results from the condition that the reflection of $p'e_1 + q'e_2 + r'e_3$ in the plane orthogonal to $pe_1 + qe_2 + re_3$ equals $p'e_1 + q'e_2 + r'e_3$ itself.

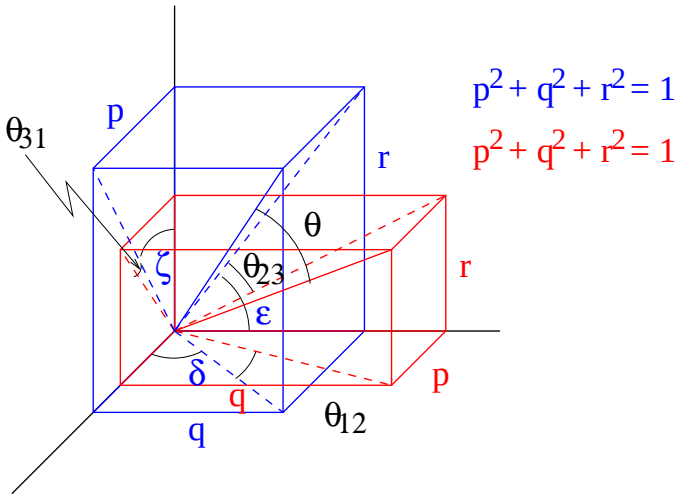
29. **Direction cosines.** The normalized p, q and r we have been using in Notes 9–11 for three dimensions are also known as direction cosines

$$p = \cos\alpha, \quad q = \cos\beta, \quad r = \cos\gamma$$

a) For line intervals, what are the angles α, β and γ?
b) What are the direction cosines in two dimensions (what must sin() be replaced by)?
c) When two line intervals, angle θ apart, are given by direction cosines, show that the sines of the projections of θ on the e_{12}, e_{23} and e_{31} planes are, respectively,

$$\sin\theta_{12} = \frac{pq' - qp'}{\sqrt{p^2 + q^2}\sqrt{p'^2 + q'^2}}, \quad \sin\theta_{23} = \frac{qr' - rq'}{\sqrt{q^2 + r^2}\sqrt{q'^2 + r'^2}}, \quad \sin\theta_{31} = \frac{rp' - pr'}{\sqrt{r^2 + p^2}\sqrt{r'^2 + p'^2}}$$

(In the figure, red is used for the primed direction cosines. Use the differences between the angles δ, ϵ and ζ shown and their red counterparts.)



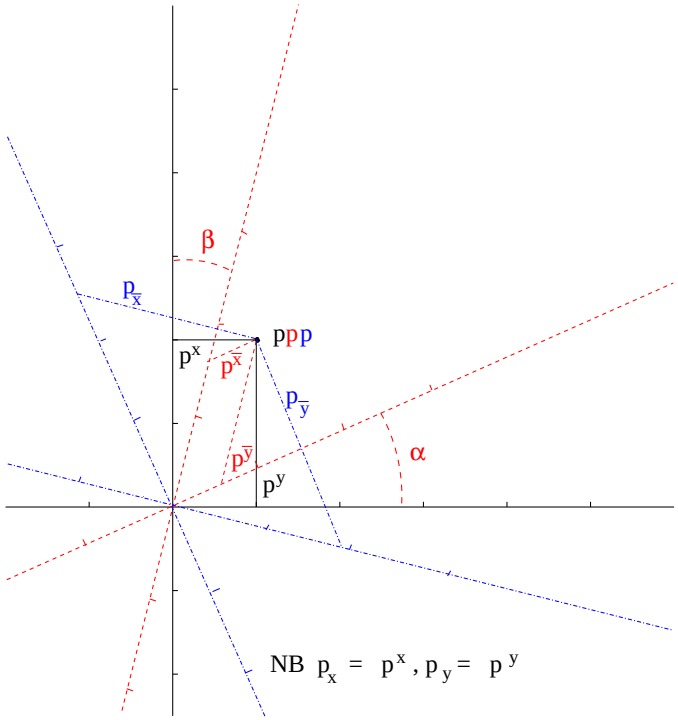
d) What are the cosines of these projections of θ?
e) Note that the redundancy of $p^2 + q^2 + r^2 = 1$ hides the signs. Why is it useful to have all three direction cosines? (All two in 2D?)

30. The Gibbs "cross product" of two vectors in 3D is defined as

$$(a, b, c) \times (a', b'c') = ((bc' - cb', ca' - ac', ab' - ba')$$

Show that this is not a vector but a tensor. Write the tensor.
Show that it is not a line interval but a plane interval. Write the plane interval.

31. **Nonorthogonal axes and tensor notation.** We've seen that the two "tensors" in Notes 3 and 4 are independent of rotations of the coordinate system. Tensor notation is intended to cope with *any* linear transformation of the axes. After the following discussion, show that the twirl of Note 4 survives *nonorthogonal* axis transformations but the height- and width-eigenvalues of Note 3 do not.
Here is a non-orthogonal axis transformation, from the black axes (solid lines) to the red (dashed lines).



a) To transform the vectors

$$\begin{pmatrix} 1 \\ 0 \end{pmatrix} \qquad \text{and} \qquad \begin{pmatrix} 0 \\ 1 \end{pmatrix}$$

to the red axes shown, persuade yourself that we would use the matrix

$$S = \frac{1}{\sqrt{c_+}} \begin{pmatrix} c_\alpha & s_\beta \\ s_\alpha & c_\beta \end{pmatrix}$$

where c_z is $\cos(z)$, s_z is $\sin(z)$ for z either α or β, and c_+ is $\cos(\alpha+\beta) = \cos(\alpha)\cos(\beta) - \sin(\alpha)\sin(\beta)$. (This latter is the determinant of the part of S written above as a matrix. We need not divide that matrix by $\sqrt{c_+}$ but to do so normalizes the matrix in a way that allows the example to illustrate a minor but significant point (see (j)).)

The first point to make is that to transform the *coordinates* describing the point $p = (p^x, p^y)$, so that the *same* point, p, is identified by new (red) coordinates, $p = (p^{\bar{x}}, p^{\bar{y}})$, we use the *inverse* of S,

$$S^{-1} = \frac{1}{\sqrt{c_+}} \begin{pmatrix} c_\beta & -s_\beta \\ -s_\alpha & c_\alpha \end{pmatrix}$$

b) Persuade yourself that this statement is true. (Think about rotations as an example, and compare finding a rotated point, p, with finding new coordinates for p under rotated *axes*.)

So

$$\begin{pmatrix} p^{\bar{x}} \\ p^{\bar{y}} \end{pmatrix} = \frac{1}{\sqrt{c_+}} \begin{pmatrix} c_\beta & -s_\beta \\ -s_\alpha & c_\alpha \end{pmatrix} \begin{pmatrix} p^x \\ p^y \end{pmatrix}$$

c) The new red coordinates are *non-orthogonal*: the red axes are not at right angles to each other. What is different about a non-orthogonal coordinate system is that invariants such as the length of a vector (say, the distance from p to the origin) or the angle between two vectors will apparently change under the transformation: calculate $\sqrt{p^{\bar{x}} \times p^{\bar{x}} + p^{\bar{y}} \times p^{\bar{y}}}$ and compare it to $\sqrt{p^x \times p^x + p^y \times p^y}$. This does not happen with orthogonal transformations such as rotations: such invariants are left safely fixed by the rotation. (Show that the two above square roots are the same if $\beta = -\alpha$ but not necessarily for other β. If your algebraic

results do not convince you, try it with the two angles drawn in the above figure: $c_\alpha = 12/13$ and $c_\beta = 24/25$. Show that the squares of the lengths are about 2.75 versus 5. Or just look at the drawing.)

So we need to think of something else. This is the first contribution of tensor theory. Because we used S^{-1} to transform p^j to $p^{\bar{k}}$ this transformation is called *contravariant*. A corresponding transformation using S^T is called *covariant*, and that is what we need. Tensor notation writes contravariant elements with superscript indices. (That is why I repeated the p^x and the p^y above to square them instead of writing p^{x2} and p^{y2}: it is best in tensor notation not to use superscript operators to denote powers.)

And covariant elements are written with subscript indices. You'll find this in the blue components, $p_{\bar{x}}$ and $p_{\bar{y}}$, in the diagram.

Only in orthogonal coordinate systems are the contravariant and the covariant components the same. Thus, in black, $p_x = p^x$ and $p_y = p^y$.

In the diagram, the blue (dot-dash lines) shows the transformation.

$$\begin{pmatrix} p_{\bar{x}} \\ p_{\bar{y}} \end{pmatrix} = \frac{1}{\sqrt{c_+}} \begin{pmatrix} c_\alpha & s_\alpha \\ s_\beta & c_\beta \end{pmatrix} \begin{pmatrix} p_x \\ p_y \end{pmatrix}$$

(d) Show that the covariant transformation of (1,0) is orthogonal to the contravariant transformation of (0,1) and vice-versa in this example.

Now we consider how to describe the invariant length of p (that is, the distance of point p from the origin). We use *both* contravariant and covariant coordinate systems. Writing the transformations as matrices and the coordinates as vectors

$$(p_{\bar{x}}, p_{\bar{y}}) \begin{pmatrix} p^{\bar{x}} \\ p^{\bar{y}} \end{pmatrix} = (p_{\bar{x}}, p_{\bar{y}}) S S^{-1} \begin{pmatrix} p^{\bar{x}} \\ p^{\bar{y}} \end{pmatrix} = (p_x, p_y) \begin{pmatrix} p^x \\ p^y \end{pmatrix} = (p^x, p^y) \begin{pmatrix} p_x \\ p_y \end{pmatrix}$$

Here I used row vector $\times$ column vector to express the sum of products (in this case, the sum of squares), and I've used the less usual way of writing the transformation $S^T \times$(column vector) as (row vector)$\times S$.

Let's look at this in terms of the indices. I'll write them out first in the conventional matrix way with all indices as subscripts.

$$\begin{aligned} \sum_j p_j p_j &= \sum_j (\sum_k p_k S_{kj})(\sum_{k'} S^{-1}_{jk'} p_{k'}) \\ &= \sum_k p_k \sum_{k'} (\sum_j S_{kj} S^{-1}_{jk'}) p_{k'} \\ &= \sum_k p_k (\sum_{k'} I_{kk'} p_{k'}) \\ &= \sum_k p_k p_k \end{aligned}$$

From the first to the second line we rearranged the order of the sums, which you should convince yourself we can always do. Then we got SS^{-1}, and this is the identity I. The identity gets rid of the sum of k' by just setting k' to k.

Because we can always rearrange summation order and because multiplication of the individual elements commutes, even though multiplication of the matrices does not, tensors introduce a second notational simplification: the *Einstein summation convention* says drop the $\sum$ signs and *just repeat indices to sum.*

Combining this with the use of superscript indices for contravariant tensors and subscript indices for covariant, we can write this argument in tensor notation. Transposes don't appear at all. But I've written a place-holding dot just to indicate which was the left (row)

and which the right (column) matrix index, in order to maintain a connection with matrix multiplication.

$$\begin{aligned} p_j p^j &= p_k S^k_{.j} (S^{-1})^j_{.k'} p^{k'} \\ &= p_k I^k_{.k'} p^{k'} \\ &= p_k p^k \end{aligned}$$

e) We saw that the second-order tensors in Notes 3 and 4 transform using both the transformation matrix and its inverse, e.g., $T \rightarrow RTR^{-1}$. Write this in tensor notation and argue that tensors that transform as T does should be written with one contravariant and one covariant index. That is why I wrote the transformation matrices, say S, above as S^k_j.
The above discussion requires that we have both contravariant and covariant components in order to compute invariants such as the inner product (length of one vector or angle between two different normalized vactors). But there is a way to find length, say, if we have only one of these sets, such as the contravariant components. We start with the length in the orthogonal system.

$$\begin{aligned} p^j p^j &= S^j_{.k} p^k S^j_{.k'} p^{k'} \\ &= p^k (S^T)^{.j}_k S^j_{.k'} p^{k'} \\ &= p^k (S^T S)_{kk'} p^{k'} \\ &= g_{kk'} p^k p^{k'} \end{aligned}$$

where $g_{kk'} \stackrel{\text{def}}{=} (S^T S)_{kk'}$ is the covariant double tensor called the *fundamental metric tensor*. g enables us to find invariants even though limiting ourselves to contravariant coordinates.
f) Calculate $g_{kk'}$ for the working example of this Excursion.
g) Show that $g_{kk'}$ is symmetric, i.e., $g_{kk'} = g_{k'k}$.
h) What is the *contravariant* fundamental metric tensor, $g^{kk'}$?
i) Show that $g_{kk'}$ applied to any tensor with a contravariant index k' (or k) *lowers* that index, making it covariant. Hint: use $p_{\bar{j}} p^{\bar{j}} = p_j p^j$.
At last, the "minor but significant point" I promised to illustrate back at the beginning of this Excursion when I introduced $1/\sqrt{c_+}$ as a normalizing factor in the example transformations.
j) Show that without that factor, S would transform (1,0) and (0,1) into vectors of the same length, 1. Then convince yourself that the scaling factor $1/\sqrt{c_+} = 1.13$ increases the distance between the unit marks along the transformed axes in the diagram.
k) The vector product $b_j x^j$ is not the only invariant. A constant, c, is always invariant, of course. And so is the 2nd-order product $g_{jk} x^j x^k$ above, as is, for *any* a_{jk}, $a_{jk} x^j x^k$.
These can be combined in a generalization of the quadratic equation for scalars $ax^2 + bx + c$ to the *general quadric*

$$a_{jk} x^j x^k + b_j x^j + c$$

What are the interpretations that can be made of the general quadric, in the sense that the quadratic can be interpreted as a parabola or, in special cases, a straight line? (Classify all the possibilities in 2D. Look into 3D: what are planes? What are lines? Show that any matrix is the sum of a symmetric and an antisymmetric matrix: what contribution to $a_{jk} x^j x^k$ is made by the antisymmetric part of a?)
l) Look up [McC57]: this Excursion prepares you for Parts I and II, where you can learn about classical geometry as an application of tensors. This is an older book but application-directed once you are over the initial hurdles—and you should now be prepared for these.
m) Without the normalizing factor $1/\sqrt{c_+}$ in the transformation, show that $g_{12} = g_{21} = s_+$ where s_+ is $\sin(\alpha+\beta) = \cos(\pi/2-(\alpha+\beta))$, the cosine of the angle between the nonorthogonal axes.

32. **Tensor Calculator I. Matrix representation.** One- and two-index tensors can be represented readily as vectors and matrices and the Einstein convention that repeated indices are summed over translates into regular matrix multiplication. Indices are either subscript (covariant) or superscript (contravariant: see previous Excursion) and raising or lowering them means (matrix) multiplication by the metric tensor or its inverse: since the metric tensor has two indices, either both down or both up, it can be included in the matrix representation employed in this Excursion. We build a MATLAB program to raise and lower indices and to multiply the matrices and vectors representing tensors.
We need a convention for the names of the matrices which distinguishes the positions of the indices, since, for instance the tensor $g_{\alpha,\beta}$ is different from $g^{\alpha,\beta}$—indeed, if it is the metric tensor, as its name implies in this example, one is the inverse of the other.
So we combine the name of the tensor with the position(s) of its index(es): $g_{\alpha,\beta}$, for example, is called `g_dd` because of the two "down" indices, and $g^{\alpha,\beta}$ is called `g_uu`.
We work with examples from [Har03, pp.423, 428]. For instance,

$$\texttt{g\_dd} = \begin{pmatrix} 2 & 1 \\ 1 & 0 \end{pmatrix}$$

a) Show that

$$\texttt{g\_uu} = \begin{pmatrix} 0 & 1 \\ 1 & -2 \end{pmatrix}$$

b) Because we must multiply a tensor by the appropriate metric tensor to raise or lower its indices,

$$t^{\alpha}_{\beta} = g^{\alpha\gamma} t_{\gamma\beta} \quad \text{or} \quad \texttt{t\_ud} = \texttt{g\_uu t\_dd}$$

and

$$t^{\beta}_{\alpha} = t_{\alpha\gamma} g^{\gamma\beta} \quad \text{or} \quad \texttt{t\_du} = \texttt{t\_dd g\_uu}$$

show that

$$\texttt{g\_ud} = \texttt{g\_uu g\_dd} = I$$

and

$$\texttt{g\_du} = \texttt{g\_dd g\_dd} = I$$

where I is the identity matrix.
c) Given, in addition to `g_dd` above, the single-index tensors

$$\begin{aligned} \texttt{a\_d} &= \begin{pmatrix} 1 \\ 0 \end{pmatrix} \\ \texttt{b\_d} &= \begin{pmatrix} 0 \\ 1 \end{pmatrix} \\ \texttt{c\_u} &= \begin{pmatrix} 1 \\ 0 \end{pmatrix} \\ \texttt{d\_u} &= \begin{pmatrix} 0 \\ 1 \end{pmatrix} \end{aligned}$$

show that

$$\begin{aligned} \texttt{a\_u} &= \begin{pmatrix} 0 \\ 1 \end{pmatrix} \\ \texttt{b\_u} &= \begin{pmatrix} 1 \\ -2 \end{pmatrix} \\ \texttt{c\_d} &= \begin{pmatrix} 2 \\ 1 \end{pmatrix} \\ \texttt{d\_d} &= \begin{pmatrix} 1 \\ 0 \end{pmatrix} \end{aligned}$$

d) Given, also in addition to `g_dd` above, the double-index tensor

$$\texttt{t\_dd} = \begin{pmatrix} 3 & 1 \\ -1 & 0 \end{pmatrix}$$

show that

$$\begin{aligned} \texttt{t\_dd} &= \begin{pmatrix} -1 & 0 \\ 5 & 1 \end{pmatrix} \\ \texttt{t\_ud} &= \begin{pmatrix} 1 & 1 \\ 0 & -1 \end{pmatrix} \\ \texttt{t\_du} &= \begin{pmatrix} 0 & -1 \\ 1 & 3 \end{pmatrix} \end{aligned}$$

We must build a table to associate each tensor name with its corresponding matrix (or vector). We can use a cell array:

```
table = {'g_dd',[2,1;1,0];'a_d',[1;0];'b_d',[0;1]; 'c_u',[1;0];'d_u',[0;1];...
         't_dd',[3,1;-1,0]};
```

and write code which will extend this table each time we calculate a new tensor.

e) Start with a lookup function

```
function [name,array,d2u,u2d] = lookup(strg,table)
```

which uses `table` passively to find in the first column of `table` the `strg` and return in `name` the closest entry and in `array` the corresponding matrix. It should furthermore return in `d2u` the positions of the indices which must be raised to convert the found `name` to the sought `strg`, and in `u2d` the positions of the indices which must be lowered. Thus, with the above `table`

```
[name,array,d2u,u2d] = lookup3('t_dd',table)
```

should return the exact match

```
name = t_dd
array =
    3 1
   −1 0
d2u = []
u2d = []
```

while

```
[name,array,d2u,u2d] = lookup3('t_ud',table)
```

should return the nearest match (exact on "`t`" and closest on the sequence of "`u`"s and "`d`"s.

```
name = t_dd
array =
   3 1
   −1 0
d2u = [1]
u2d = []
```

and

```
[name,array,d2u,u2d] = lookup3('t_uu',table)
```

will return the same except

```
d2u = [1 2]
```

and finally

```
[name,array,d2u,u2d] = lookup3('x_dd',table)
```

reports that no tensor starting "`x`" is in `table`

```
name = ''
array = Inf
d2u = []
```

```
u2d = []
```

f) Now build the tensor calculator itself

```
function [answer,newTable] = tensorCalc(query,table)
```

which applies `lookup(query,table)` to give the suitably modified matrix as `answer` and also enters the new name and matrix into `newTable`. (You can go one further and let `query` be a sequence of `strg`s for `lookup()`, and, if the resulting matrices have suitably matching sizes, add at the end of `answer` the matrix product of the whole lot.) Thus,

```
[answer,newTable] = tensorCalc3('t_uu','c_d',table)
answer = [2x2 double] [2x1 double] [2x1 double]
newTable =
'g_dd' [2x2 double]
'a_d' [2x1 double]
'b_d' [2x1 double]
'c_u' [2x1 double]
'd_u' [2x1 double]
't_dd' [2x2 double]
'g_uu' [2x2 double]
't_uu' [2x2 double]
'c_d' [2x1 double]
answer:
ans =
   0 -1
   1 3
ans =
   2
   1
ans =
   -1
   5
```

Tensor notation is more general than matrices, so a matrix representation will not capture the full capabilities of tensors. For example, the above does not support dot product of two vectors, since vectors are all represented as columns (this could be patched, of course). It does not support contraction since that is expressed by repeated indices, which our representation is not capable of. And tensors of three or more indices will of course be awkward to express as conventional matrix multiplication.

33. **Tensor Calculator II. Relational representation.** A suitable generalization of matrices which can represent general tensors are relations, a computer data structure proposed by E. F. Codd in 1970 to rationalize databases on secondary storage but applicable much more generally and to matrices in particular. We must switch from MATLAB to the language Aldat [Mer07].

A *relation* is a set of "n-tuples". It is a set in the mathematical sense, that it has no duplicate elements and that the order of the elements does not matter. For example, $\{a, b, b\}$ is not a set; $\{a, b, c\}$ is the same set as $\{a, c, b\}$ or as any permutation of these three letters. The order-independence of sets is a powerful abstraction which reduces the 3! (3-factorial) possible permutations of this example to a single entity.

An *n-tuple* is a collection of n items for which order *does* matter, either absolutely, or relative to some pre-ordered list of names called *attributes*.

Examples of relations representing vectors and matrices are

$$a_d = \begin{pmatrix} 1 \\ 0 \end{pmatrix} \qquad g_{uu} = \begin{pmatrix} 0 & 1 \\ 1 & -2 \end{pmatrix} \qquad t_{dd} = \begin{pmatrix} -1 & 0 \\ 5 & 1 \end{pmatrix}$$

```
a_d(i  v)
   1  1
```

```
g_uu(i  j  v)
     1  2  1
     2  1  1
     2  2 -2
```

```
t_dd(i  j  v)
     1  1  3
     1  2  1
     2  1 -1
```

The indices are stored explicitly, as attributes named `i, j`, etc., and the value of each element has its own attribute named `v`. The relations each have a name, before the parentheses naming the attributes. Relation `a_d` is a "binary" relation: its elements being 2-tuples (or duples or pairs). Relations `g_uu` and `t_dd` are "ternary" relations, with 3-tuples (or triples) as elements, The data follows in columns below the names in each heading line. (This particular way of showing relations is called the "tabular representation".)
Here is a 3-index tensor, the "alternating tensor". Its elements are 4-tuples or quadruples.

```
alt_uuu(i  j  k  v)
        1  2  3  1
        1  3  2 -1
        2  3  1  1
        2  1  3 -1
        3  1  2  1
        3  2  1 -1
```

Note that zero entries are conveniently omitted: the relational representation of ordinary matrices is especially attractive when the matrices are "sparse", i.e., have a large proportion of zeros. The alternating tensor in particular, although a three-index array, not an ordinary matrix, is sparse: it has three 1s, three -1s and $3 \times 3 \times 3 - 6 =$ twenty-one 0s.
Relations are manipulated by accompanying operations of the *relational algebra* and of the *attribute algebra*. The former creates new relations from existing relations. The latter creates new attributes from existing attributes.
The relational algebra supports a *renaming assignment*, a *natural join* (a "binary" operation on two relations), and a *T-selector* (a "unary" operation on one relation).
Here are assignments to rename attributes in `t_dd` and `g_uu`.

```
g_uu'[i,joinon,vg <- i,j,v]g_uu;
t_dd'[joinon,j,vv <- i,j,v]t_dd;
```

The results are

```
g_uu'(i  joinon  vg)
      1     2     1
      2     1     1
      2     2    -2
```

```
t_dd'(joinon  j  vv)
           1  1   3
           1  2   1
           2  1  -1
```

The data are the same, but the attributes have been renamed. (For a subtle reason, this operation is considered part of the relational algebra, not of the attribute algebra.) Note that the prime character, `'`, is just part of the relation name in Aldat, not an operator as in MATLAB.
The *natural join* combines two relations on their common attribute(s) by merging all tuples from the first relation with all tuples from the second relation that match on values of that (those) shared attribute(s). Here is the result of joining the two relations we've just renamed so that they *have* a common attribute, `joinon`.

```
g_uu' natjoin t_dd'
(i  joinon  j   vg  vv)  vgv   v
 2    1     1    1    3    3    5
 2    1     2    1    1    1    1
 1    2     1    1   −1   −1   −1
 2    2     1   −2   −1    2    5
```

(Ignore for the moment the "virtual attributes" `vgv` and `v`, which will be defined when we reach the attribute algebra, below.)
Examine carefully how the `(2, 1, 1)` tuple from `g_uu'` is duplicated because two tuples from `t_dd'` share its `joinon` value of `1`, and how the `(2, 1, −1)` tuple from `t_dd'` is duplicated because two tuples from `g_uu'` share its `joinon` value of `2`. The above joined relation is displayed with the values of the common attribute, `joinon`, grouped to reveal this merge.
a) Show that 33 tuples result from the natural join of relation `R(A,B)`, which has 4 tuples with `B = 1`, 5 tuples with `B = 2`, and 3 tuples with `B = 3`, with relation `S(B,C)`, which has 3 tuples with `B = 2`, 6 tuples with `B = 3` and 2 tuples with `B = 4`.
The *T-selector* combines *selecting* certain tuples from a relation according to some condition on the values of its attributes, with *projecting* specified attributes from the result. Here is an example which selects the value `1` from `joinon` and then projects on the attributes other than `joinon`.

```
[i,j,vg,vv] where joinon=1 in (g_uu' natjoin t_dd')
(i  j  vg  vv)
 2  1   1   3
 2  2   1   1
```

(The projection component must remove duplicates if any result from the disappearance of attribute(s).)
b) Show that only one tuple results from: `[i] where joinon/abs(vg)=1 in g_uu'`

The *attribute algebra* creates new attributes from old, independently of any relational context. So the results are *virtual attributes*, potentially available to any relation which has all the antecedent attributes. Thus

```
let vgv be vg*vv
```

defines the product of two attributes. Since these attributes are both in the relation which is the natural join above, `g_uu' natjoin t_dd'`, the result, `vgv`, is meaningful in the context of this join and so can be shown in connection with it, as we did above. It is, however, not an attribute of this join, although it could be *actualized* in a new relation which is a projection of it.

```
gt <- [i,joinon,j,vg,vv,vgv] in (g_uu' natjoin t_dd')
```

```
gt(i  joinon  j   vg   vv  vgv)   v
   1    2     1    1   −1   −1   −1
   2    1     1    1    3    3    5
   2    2     1   −2   −1    2    5
   2    1     2    1    1    1    1
```

Again, ignore for the moment the virtual attribute `v`.
(Two small syntactical comments: `<-` is the *assignment* operator of the relational algebra which, without renaming attributes, creates a new named relation; omitting the `where` syntax gives pure projection, and a similar omission gives pure selection, as special cases of the

T-selector.)
Projection, and relational algebra expressions in general, may be used to actualize virtual attributes created by the attribute algebra.
c) Show that `vgv` is as shown in relation `gt` above.
The second operator of the attribute algebra that we must discuss is *aggregation*. This aggregates values within an attribute, creating a new attribute which is also virtual until actualized. For example, the virtual attribute `v` shown above beside `vgv` associated with the natural join `g_uu' natjoin t_dd'`, is the aggregate sum over tuples sharing common values of attributes `i`and `j`.

```
let v be equiv + of vv*vg by i,j
```

This groups tuples according to common values of `i` and `j` and sums the values of `vg*vv` within these groupings. The tuples in `gt` above are ordered to reveal these groups. Note that `vg*vv` is an unnamed virtual attribute. We could instead have used the name, `vgv`, we created above.
(We could invent syntax specialized for tensor index sums

```
let v be agg + of vg*vv on joinon
```

where the attributes `i` and `j` are the *complements* in, say, relation `gt`, of `vg, vv` and the `on` attribute `joinon`. But the syntax referring to equivalence classes of tuples is clearer to work with in the relational representation.)
If the Aldat programmer is careful in actualizing the virtual attribute `v` there will be no conflict with the attribute `v` which already actually appears in relations `g_uu` and `t_dd`. In a projection, or any relational algebra operation, virtual attribute definitions in the attribute algebra are ignored if an attribute of that name is already actual in any relation in that relational algebra expression.
d) Confirm that the three different values of v above are those that appear in the matrix product $g_{uu} \times t_{dd}$.
This choice of relational and attribute algebra operations permits us to multiply ordinary matrices. Here is the full code, starting with the original relational representations `g_uu(i,j,v)` and `t_dd(i,j,v)`.

```
t_dd'[joinon,j,vv <- i,j,v]t_dd;
g_uu'[joinon,i,vg <- i,j,v]g_uu;
let v be equiv + of vv*vg by i,j;
t_ud <- [i,j,v] in (t_dd' natjoin g_uu');
```

(Note that g_{uu} is symmetrical, so we may swap the first two attributes of its relational representation.)
e) Show that `t_ud` obtained above is the relational representation for the matrix t_{ud}.
f) Write Aldat code, analogous to the above, to calculate `t_du` and `t_uu`. Show that the latter can be derived in two different ways.
g) Represent `a_d, b_d, c_u, d_u` and `g_dd` from the previous Excursion as relations and write code to find `a_u, b_u, c_d, d_d`. (Calculating `g_dd` as the inverse of `g_uu` using, say, Gaussian elimination, can be done in Aldat, but there is no builtin operator to do this.)
The dot product, `a.b`, can also be coded relationally.

```
a_u'[joinon,vv <- i,v]a_u;
b_d'[joinon,vw <- i,v]b_d;
let v be red + of vv*vw;
ab <- [v] in (a_u' natjoin b_d');
```

The new attribute algebra operator, `red`, can be thought of as `equiv` followed by an empty `by`-list: there are no groupings and the sum is over the whole relation. (The syntax `let v be agg + of vv*vw on joinon` would also be valid here.)
h) Use `red +` to find the *trace* of, say, `t_ud`.
So far we have not advanced much beyond matrices and MATLAB. Now let's go to three

dimensions, with

$$g\_dd = \begin{pmatrix} 2 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & -1 \end{pmatrix}$$

(whose *determinant* $g = 1$) and the three-index *alternating tensor*

$$\frac{1}{\sqrt{g}}\epsilon^{ijk}$$

where $\epsilon^{ijk} = 1$ if (i, j, k) is an even permutation of (1,2,3), $\epsilon^{ijk} = -1$ if (i, j, k) is an odd permutation of (1,2,3), and otherwise zero. Here is the alternating tensor explicitly in Aldat

```
relation alt_uuu(i,j,k,v) <-
 {(1,2,3,1),(2,1,3,-1),(2,3,1,1),(3,2,1,-1),(3,1,2,1),(1,3,2,-1)};
```

(incidentally showing how to declare and how to initialize a relation). This was displayed at the beginning of this Excursion (with a different ordering of the tuples).
To lower the first index, we need the relational representation of `g_dd` and the Aldat code

```
alt_uuu'[joinon,j,k,va <- i,j,k,v]alt_uuu;
g_dd'[joinon,i,vg <- i,j,v]g_dd;
let v be equiv + of va*vg by i,j,k;
alt_duu <- [i,j,k,v] where v!=0 in (alt_uuu' natjoin g_dd');
```

giving

```
alt_duu(i  j  k  v)
        1  1  3  -1
        1  2  3   2
        1  3  1   1
        1  3  2  -2
        2  2  3   1
        2  3  2  -1
        3  1  2  -1
        3  2  1   1
```

i) Calculate the remaining six versions of `alt` and show (the following uses array, not relational, notation)

- $\texttt{alt}_{udu}(i, j, k) = \texttt{alt}_{duu}(k, i, j)$ and $\texttt{alt}_{uud}(i, j, k) = \texttt{alt}_{duu}(j, k, i)$;
- $\texttt{alt}_{dud}(i, j, k) = \texttt{alt}_{udd}(k, i, j)$ and $\texttt{alt}_{ddu}(i, j, k) = \texttt{alt}_{udd}(j, k, i)$;
- $\texttt{alt}_{duu}(i, j, k) = -\texttt{alt}_{duu}(i, k, j)$, $\texttt{alt}_{udu}(i, j, k) = -\texttt{alt}_{udu}(i, k, j)$ and $\texttt{alt}_{uud}(i, j, k) = -\texttt{alt}_{uud}(i, k, j)$;
- $\texttt{alt}_{udd}(i, j, k) = -\texttt{alt}_{udd}(i, k, j)$, $\texttt{alt}_{dud}(i, j, k) = -\texttt{alt}_{dud}(i, k, j)$ and $\texttt{alt}_{ddu}(i, j, k) = -\texttt{alt}_{ddu}(i, k, j)$; and
- $\texttt{alt}_{ddd} = \texttt{alt}_{uuu}$.

How many different ways are there to calculate, say, `alt_udd` from this starting point? `alt_ddd`?
j) Find a way to write out these alternating tensors as three-dimensional "matrices". Do this in a way to show the antisymmetries of each of the eight versions.
To run these programs other than by hand you must install a copy of Aldat on your computer: /citealdat.
Although Aldat is written in Java it is designed to run under UNIX-like operating systems.
It is possible, if convoluted given the lack of polymorphism in nested relations, to write Aldat code analogous to `tensorCalc` in the previous Excursion.

34. Look up H. S. M. Coxeter's *Regular Polytopes* [Cox63] and use the interval algebra to construct higher-dimensional versions of the tetrahedron, cube and octahedron.

35. How might we use the interval algebra to describe a shear operation?

36. Look up William Kingdon Clifford, 1845–1879, and describe his role in creating the interval algebra. (It is really called the Clifford algebra, or sometimes the geometric algebra.)

37. Look up Josiah Willard Gibbs, 1839–1903, and his vector analysis.

38. Look up Sir William Rowan Hamilton, 1805–1865, and his "quaternions". What mental block stumped him for a long time? How did he misinterpret what he invented, and how do quaternions relate to 3D interval algebra? (see [Alt92].)

39. How do the Pauli matrices (Week 6) relate to 3D interval algebra?

40. Why is the number of basic elements of d-dimensional interval algebra equal to 2^d?

41. Survey the usage of the phrase "real world" and distinguish a legitimate usage from a put-down of academics.

42. Any part of the Preliminary Notes that needs working through.

References

[Alt92] Simon L. Altmann. *Icons and Symmetries*. Clarendon Press, Oxford, 1992.

[Cox63] H. S. M. Coxeter. *Regular Polytopes*. The MacMillan Company, Collier-MacMillan Ltd, New York, London, 1963. 2nd ed.

[Har03] James B Hartle. *Gravity: An Introduction to Einstein's General Relativity*. Addison Wesley, San Francisco, 2003.

[McC57] A. J. McConnell. *Applications of Tensor Analysis*. Dover Publictions, Inc., New York, 1957. Originally *Applications of the Absolute Differential Calculus*, 1931.

[Mer84] T. H. Merrett. *Relational Information Systems*. Reston Publishing Co., Reston, VA., 1984.

[Mer99] T. H. Merrett. Relational information systems. (revisions of [Mer84]):
Data structures for secondary storage: http://www.cs.mcgill.ca/∼tim/cs420
Database programming: http://www.cs.mcgill.ca/∼tim/cs612, 1999.

[Mer07] T. H. Merrett. Aldat: a retrospective on a work in progress. *Information Systems*, 32(4):505–44, March 2007.