On their wishlist are facilities along the line of the ESTIMATE and LSMEANS statements available in many SAS procedures (including GLM, MIXED and GENMOD) for easy specification of various contrasts (LSMEANS is sometimes also denoted "population means"). While LSMEANS are often misused (and certainly misinterpreted) such facilities would be nice to have in R. I would like to encourage anyone who has implemented such facilities in a reasonable level of generality to come forward.

Søren Højsgaard
Statistics and Decision Analysis Unit
Department of Genetics and Biotechnology
Danish Institute of Agricultural Sciences
Tjele, Denmark
sorenh@agrsci.dk

Normed division algebras with R: Introducing the onion package

The eccentric cousin and the crazy old uncle

by Robin K. S. Hankin

Preface

An *algebra* is a vector space V over a field (here the real numbers) in which the vectors may be multiplied. In addition to the usual vector space axioms, one requires, for any $\lambda, \mu \in \mathbb{R}$ and $\mathbf{x}, \mathbf{y}, \mathbf{z} \in V$:

- $\mathbf{x}(\lambda\mathbf{y} + \mu\mathbf{z}) = \lambda\mathbf{xy} + \mu\mathbf{xz}$
- $(\lambda\mathbf{y} + \mu\mathbf{z})\mathbf{x} = \lambda\mathbf{yx} + \mu\mathbf{zx}$

where multiplication is denoted by juxtaposition; note the absence of associativity. A *division algebra* is a nontrivial algebra in which division is possible: for any $\mathbf{a} \in V$ and any nonzero $\mathbf{b} \in V$, there exists precisely one element $\mathbf{x}$ with $\mathbf{a} = \mathbf{bx}$ and precisely one element $\mathbf{y}$ with $\mathbf{a} = \mathbf{yb}$. A *normed division algebra* is a division algebra with a norm $||\cdot||$ satisfying $||\mathbf{xy}|| = ||\mathbf{x}||\;||\mathbf{y}||$.

There are precisely four normed division algebras: the reals themselves ($\mathbb{R}$), the complex numbers ($\mathbb{C}$), the quaternions ($\mathbb{H}$) and the octonions ($\mathbb{O}$); the generic term is "onion", although the term includes other algebras such as the sedenions.

The R programming language (Development Core Team, 2004) is well-equipped to deal with the first two: here, I introduce the **onion** package, which provides some functionality for the quaternions and the octonions, and illustrate these interesting algebras using numerical examples.

Introduction

Historically, the complex numbers arose from a number of independent lines of inquiry and our current understanding of them (viz $z = x + iy$; the Argand plane) developed over the eighteenth and nineteenth centuries.

Hamilton was one of many mathematicians to attempt to extend the complex numbers to a third dimension and discover what would in our terminology be a three dimensional normed division algebra. We now know that no such thing exists: division algebras all have dimension 2^n for n some non-negative integer.

Hamilton came upon the multiplicative structure of the quaternions in a now-famous flash of inspiration on 16th October 1843: the quaternions are obtained by adding the elements i, j, and k to the real numbers and requiring that

$$i^2 = j^2 = k^2 = ijk = -1. \quad (1)$$

A general quaternion is thus written $a + bi + cj + dk$ with a, b, c, d being real numbers; complex arithmetic is recovered if $c = d = 0$. Hamilton's relations above, together with distributivity and associativity, yield the full multiplication table and the quaternions are the unique four dimensional normed division algebra.

However, Hamilton's scheme was controversial as his multiplication was noncommutative: a shocking suggestion at the time. Today we recognize many more noncommutative operations (such as matrix multiplication), but even Hamilton had difficulty convincing his contemporaries that such operations were consistent, let alone worth studying.

The octonions

The fourth and final normed division algebra is that of the octonions. These were discovered around 1844 and are an eight-dimensional algebra over the reals. The full multiplication table is given by Baez (2001).

Package onion in use

A good place to start is function `rquat()`, which returns a quaternionic vector of a specified length, whose elements are random small integers:

```
> library(onion)
> x <- rquat(5)
> names(x) <- letters[1:5]
> print(x)

   a b c d e
Re 4 3 2 1 3
i  3 2 3 4 1
j  2 4 2 3 1
k  3 3 4 4 4
```

This quaternionic vector, of length 5, is of the form of a four-row matrix. The rownames are the standard basis for quaternions, and their entries may be manipulated as expected; for example, we may set component k to be component j plus 10:

```
> k(x) <- j(x) + 10
> x

    a  b  c  d  e
Re  4  3  2  1  3
i   3  2  3  4  1
j   2  4  2  3  1
k  12 14 12 13 11
```

Quaternionic vectors behave largely as one might expect. For example, we may concatenate a with a basis quaternion (the four bases are represented in the package by H1, Hi, Hj, and Hk) and multiply the result by the first element:

```
> c(x, Hk) * x[1]

      a    b    c    d    e
Re -141 -170 -149 -170 -125 -12
i    24   -3   18    9   23   2
j    16    4   12   23  -11  -3
k    96  100   72   65   81   4
```

And indeed we may explicitly verify that quaternionic multiplication is not commutative using the commutator() function, returning $xy - yx$:

```
> y <- rquat(5)
> commutator(x, y)

     a   b   c   d   e
Re   0   0   0   0   0
i   20  12  32  98  62
j  -42 -48 -24 -44 -40
k    2  12  -4 -20  -2
```

It is possible to verify that quaternionic multiplication is associative using function associator(x,y,z) which returns $(xy)z - x(yz)$ and is thus identically zero for associative operators:

```
> associator(x, y, rquat(5))

   a b c d e
Re 0 0 0 0 0
i  0 0 0 0 0
j  0 0 0 0 0
k  0 0 0 0 0
```

Compare the octonions, which are not associative:

```
> associator(roct(3), roct(3),
+     roct(3))

    [1]  [2]  [3]
Re    0    0    0
i   472 -408  242
j  -158  112  -66
k  -106 -408   84
l  -492  128  506
il  506   54 -700
jl  250  298  610
kl -160  518 -768
```

Many transcendental functions operate on onions, via a reasonably straightforward generalization of the complex case. Consider the square root, defined as $\exp(\log(x)/2)$. That this obeys similar rules to the usual square root may be demonstrated for octonions by calculating $\sqrt{x}\sqrt{x} - x$, and showing that its modulus is small:

```
> x <- roct(3)
> Mod(sqrt(x) * sqrt(x) - x)

[1] 7.601583e-15 5.291934e-15
[3] 4.110825e-15
```

showing acceptable accuracy in this context. However, many identities that are true for the real or complex case fail for quaternions or octonions; for example, although $\log(x^2) = 2\log(x)$, it is not generally the case that $\log(xy) = \log(x) + \log(y)$, as we may verify numerically:

```
> x <- rquat(3)
> y <- rquat(3)
> Mod(log(x * x) - 2 * log(x))

[1] 0.000000e+00 3.845925e-16
[3] 0.000000e+00

> Mod(log(x * y) - log(x) - log(y))

[1] 0.0000000 0.6770686 1.1158104
```

Practical applications

I now show the quaternions in use: they can be used to rotate an object in 3D space in an elegant and natural way. If we wish to rotate vector $\overline{v}$, considered to be a triple of real numbers, we first identify its three components with the imaginary part of a quaternion with zero real part (i.e., $\mathbf{v} = 0 + \overline{v}_1 i + \overline{v}_2 j + \overline{v}_3 k$). Then the transformation defined by

$$\mathbf{v} \longrightarrow \mathbf{v}' = \mathbf{z}\mathbf{v}\mathbf{z}^{-1}$$

where $\mathbf{z} \in \mathbb{H}$, is a rotation[1]. Note that the noncommutativity of the quaternions means that the mapping is not necessarily the identity. This scheme may be used to produce Figure 1.

```
> data(bunny)
> par(mfrow = c(2, 2))
> par(mai = rep(0.1, 4))
> p3d(rotate(bunny, H1), box = FALSE,
+     d0 = 0.4, r = 1e+06, h = NULL,
+     pch = 16, cex = 0.3, theta = 0,
+     phi = 90, main = "z=1 (identity)")
> p3d(rotate(bunny, Hi), box = FALSE,
+     d0 = 0.4, r = 1e+06, h = NULL,
+     pch = 16, cex = 0.3, theta = 0,
+     phi = 90, main = "z=i")
> p3d(rotate(bunny, H1 - Hj),
+     box = FALSE, d0 = 0.4,
+     r = 1e+06, h = NULL, pch = 16,
+     cex = 0.3, theta = 0, phi = 90,
+     main = "z=1-i")
> p3d(rotate(bunny, Hall), box = FALSE,
+     d0 = 0.4, r = 1e+06, h = NULL,
+     pch = 16, cex = 0.3, theta = 0,
+     phi = 90, main = "z=1+i+j+k")
```

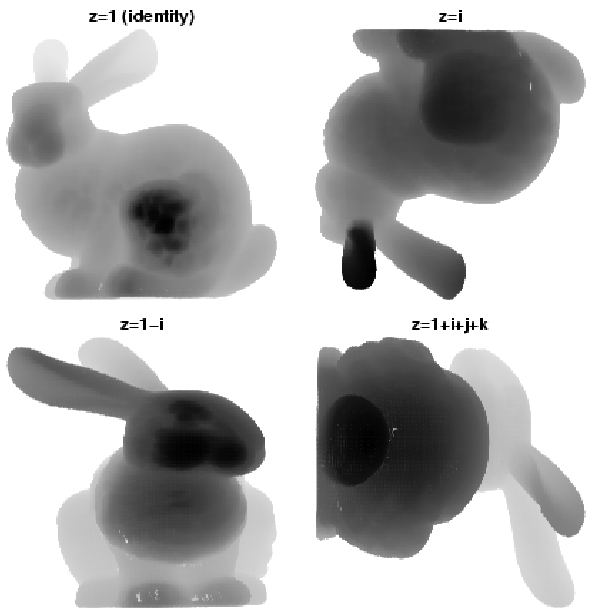

Figure 1: The Stanford Bunny (Turk, 2005) in various rotations, plotted using `p3d()`. Depth cue is via progressive greying out of points further from the eye, as though viewed through a mist

Translating between quaternionic rotation and (say) the equivalent orthogonal matrix is straightforward.

Conclusions

Quaternions and octonions are interesting and instructive examples of normed division algebras. Quaternions are a natural and efficient representation of three dimensional rotations and this paper includes a brief illustration of their use in this context.

Octonions' applicability to physics is currently unclear, but a steady trickle of papers has appeared in which the octonions are shown to be related to spinor geometry and quantum mechanics (see Baez, 2001, and references therein). Reading the literature, one cannot help feeling that octonions will eventually reveal some deep physical truth. When this happens, the **onion** package ensures that R will be ready to play its part.

Acknowledgments

I would like to acknowledge the many stimulating and helpful comments made by the R-help list over the years.

Bibliography

J. C. Baez. The octonions. *Bulletin of the American Mathematical Society*, 39(5):145–205, 2001.

R Development Core Team. *R: A language and environment for statistical computing*. R Foundation for Statistical Computing, Vienna, Austria, 2004. URL http://www.R-project.org. ISBN 3-900051-07-0.

G. Turk. The Stanford bunny, 2005. Stanford University Computer Graphics Laboratory, http://graphics.stanford.edu/data/3Dscanrep/.

Robin Hankin
National Oceanography Centre, Southampton
European Way
Southampton
United Kingdom
SO14 3ZH
r.hankin@noc.soton.ac.uk

[1]The real part of $\mathbf{v}'$ is again zero and thus may be interpreted as a three-vector. It is straightforward to prove that $\overline{v} \cdot \overline{w} = \overline{v'} \cdot \overline{w'}$. In R idiom, one would numerically verify these identities by evaluating `d <- (z*v/z) %.% (z*w/z) - v %.% w` (where `v`,`w`,`z` are quaternions) and observing that `Mod(d)` is small.

Resistor networks R: Introducing the ResistorArray package

Electrical properties of resistor networks

by Robin K. S. Hankin

Introduction

Many elementary physics courses show how resistors combine in series and parallel (Figure 1); the equations are

$$R_{\text{series}} = R_1 + R_2 \tag{1}$$

$$R_{\text{parallel}} = \frac{1}{{R_1}^{-1} + {R_2}^{-1}}. \tag{2}$$

However, these rules break down for many systems such as the Wheatstone bridge (Figure 2); the reader who doubts this should attempt to apply equations 1 and 2 and find the resistance between points A and B in the general case.

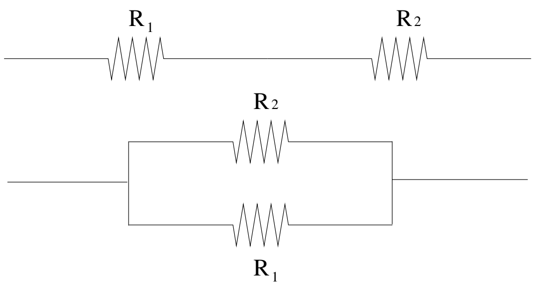

Figure 1: Two resistors in series (top) and parallel (bottom)

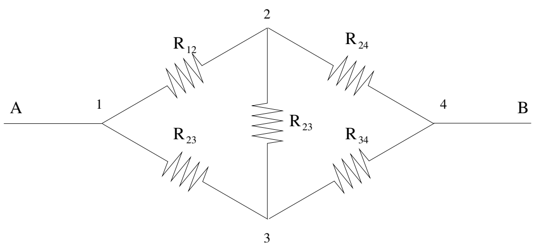

Figure 2: The Wheatstone bridge

This paper introduces **ResistorArray** (Hankin, 2005), an R (R Development Core Team, 2005) package to determine resistances and other electrical properties of such networks.

Although this paper uses the language of electrical engineering, the problem considered is clearly very general: many systems are composed of isolated nodes between which some quantity flows and the steady states of such systems are generally of interest.

Package **ResistorArray** has been applied to such diverse problems as the diffusion of nutrients among fungal hyphae networks, the propagation of salinity between (moored) oceanographical buoys, and hydraulic systems such as networks of sewage pumps.

Matrix formulation

The general problem of determining the resistance between two nodes of a resistor network requires matrix techniques[1].

Consider a network of n nodes, with node i connected to node j by a resistor of resistance R_{ij}. Then the network has a “conductance matrix” $\mathcal{L}$ with

$$\mathcal{L}_{ij} = \begin{cases} -1/R_{ij} & \text{if } i \neq j \\ \sum_{k:k\neq j} 1/R_{kj} & \text{if } i = j. \end{cases} \tag{3}$$

Thus $\mathcal{L}$ is a symmetrical matrix, whose row sums and column sums are zero (and therefore is singular).

Then the analogue of Ohm's law (viz $V = IR$) would be

$$\mathcal{L}\mathbf{v} = \mathbf{i} \tag{4}$$

where $\mathbf{v} = (v_1, \ldots, v_n)$ is a vector of potentials and $\mathbf{i} = (i_1, \ldots, i_n)$ is a vector of currents; here i_p is the current flow in to node p. Equation 4 is thus a restatement of the fact that charge does not accumulate at any node.

Each node of the circuit may *either* be fed a known current[2] and we have to calculate its potential; *or* it is maintained at a known potential and we have to calculate the current flux into that node to maintain that potential. There are thus n unknowns altogether.

Thus some elements of $\mathbf{v}$ and $\mathbf{i}$ are known and some are unknown. Without loss of generality, we may partition these vectors into known and unknown parts: $\mathbf{v} = (\mathbf{v}^{k\prime}, \mathbf{v}^{u\prime})'$ and $\mathbf{i} = (\mathbf{i}^{u\prime}, \mathbf{i}^{k\prime})'$. Thus the known elements of $\mathbf{v}$ are $\mathbf{v}^k = (v_1, \ldots, v_p)'$: these would correspond to nodes that are maintained at a specified potential; the other elements $\mathbf{v}^u = (v_{p+1}, \ldots, v_n)'$ correspond to nodes that are at an unknown potential that we have to calculate.

The current vector $\mathbf{i}$ may similarly be decomposed, but in a conjugate fashion; thus elements $\mathbf{i}^u = (i_1, \ldots, i_p)'$ correspond to nodes that require a certain, unknown, current to be fed into them to

[1]The method described here is a standard application of matrix algebra, but does not appear to be described in the literature. Many books and papers, such as Doyle and Snell (1984), allude to it; but a formal description does not appear to have been published.

[2]This would include a zero current: in the case of the Wheatstone bridge of Figure 2, nodes 2 and 3 have zero current flux so i_2 and i_3 are known and set to zero.