TRAGULA
Volume 0, Issue 0, 0000
doi: 10.5555/12345678
https://github.com/RobinHankin/evitaicossa

Antiassociative algebra in R: introducing the evitaicossa package

Robin K. S. Hankin
University of Stirling

Abstract

In this short article I introduce the **evitaicossa** package which provides functionality for antiassociative algebras in the R programming language; it is available on CRAN at https://CRAN.R-project.org/package=evitaicossa. To cite this article or the package, use Hankin (2024).

Keywords: Antiassociative algebra.

This document has not been peer-reviewed and is not an accepted Tragula publication. It is intended as an example of the style and substance appropriate for submission to Tragula.

1. Introduction

Here I introduce the **evitaicossa** R package for antiassociative algebras. An *algebra* is a vector space in which the vectors possess a well-behaved bilinear product. Formally, a vector space is a set V of vectors which form an Abelian group under addition and also satisfy the following axioms:

- Compatibility, $a(b\mathbf{v}) = (ab)\mathbf{v}$
- Identity, $1\mathbf{v} = \mathbf{v}$
- Distributivity of vector addition, $a(\mathbf{u}+\mathbf{v}) = a\mathbf{u} + a\mathbf{v}$
- Distributivity of field addition, $(a+b)\mathbf{u} = a\mathbf{u} + b\mathbf{u}$

Above, $\mathbf{u}, \mathbf{v} \in V$ are vectors, a, b are scalars [here the real numbers], and 1 is the multiplicative identity. We also require a bilinear vector product, mapping pairs of vectors to vectors; vector multiplication is denoted using juxtaposition, as in $\mathbf{uv}$, which satisfies the following axioms:

- Right distributivity, $(\mathbf{u}+\mathbf{v})\mathbf{w} = \mathbf{uw} + \mathbf{uw}$

- Left distributivity, $\mathbf{w}(\mathbf{u}+\mathbf{v}) = \mathbf{wu}+\mathbf{wv}$
- Compatibility, $(a\mathbf{u})(b\mathbf{v}) = (ab)(\mathbf{uv})$

Note the absence of commutativity and associativity. Associative algebras seem to be the most common, and examples would include multivariate polynomials (Hankin 2022c,f), Clifford algebras (Hankin 2022a), Weyl algebras (Hankin 2022e), and free algebras (Hankin 2022d). Non-associative algebras would include the octonions (Hankin 2006) and Jordan algebras (Hankin 2023). Here I consider *antiassociative* algebras in which the usual associativity relation $\mathbf{u}(\mathbf{vw}) = (\mathbf{uv})\mathbf{w}$ is replaced by the relation $\mathbf{u}(\mathbf{vw}) = -(\mathbf{uv})\mathbf{w}$.

2. Antiassociative algebras

Algebras satisfying $\mathbf{u}(\mathbf{vw}) = -(\mathbf{uv})\mathbf{w}$ exhibit some startling behaviour. Firstly, in the vector space there are no scalars except for $0 \in \mathbb{R}$. Proof: for any $x \in \mathbb{R}$, we have $x^3 = x(xx) = -(xx)x = -x^3$; thus $x^3 = -x^3$, so $x = 0$. Secondly, antiassociative algebras are nilpotent of order 4:

$$(\mathbf{ab})(\mathbf{cd}) = -\mathbf{a}(\mathbf{b}(\mathbf{cd})) = \mathbf{a}((\mathbf{bc})\mathbf{d}) = -(\mathbf{a}(\mathbf{bc}))\mathbf{d} = ((\mathbf{ab})\mathbf{c})\mathbf{d} = -(\mathbf{ab})(\mathbf{cd})$$

We see that $(\mathbf{ab})(\mathbf{cd}) = -(\mathbf{ab})(\mathbf{cd})$ so $\mathbf{abcd}$ (however bracketed) must be zero.

2.1. The free antiassociative algebra

We consider vector spaces generated by a finite alphabet of symbols $\mathbf{x}_1, \ldots, \mathbf{x}_n$. These will be denoted generally by a single letter, as in $\mathbf{a}, \mathbf{b}, \ldots, \mathbf{z}$. We now consider the algebra spanned by products of linear combinations of these symbols, subject only to the axioms of an algebra [and the antiassociative relation $\mathbf{u}(\mathbf{vw}) = -(\mathbf{uv})\mathbf{w}$]. Given an alphabet $\mathbf{x}_1, \ldots, \mathbf{x}_n$, the general form of an element of an antiassociative algebra will be

$$\sum_i \alpha_i \mathbf{x}_i + \sum_{i,j} \alpha_{ij} \mathbf{x}_i \mathbf{x}_j + \sum_{i,j,k} \alpha_{ijk} (\mathbf{x}_i \mathbf{x}_j) \mathbf{x}_k$$

(see (Remm 2024) for a proof, but note that she uses $\mathbf{x}_i(\mathbf{x}_j\mathbf{x}_k)$ rather than $(\mathbf{x}_i\mathbf{x}_j)\mathbf{x}_k$ for the triple products; a brief discussion is given in the appendix). In the package, the components of the first term $\sum_i \alpha_i \mathbf{x}_i$ are known as "single-symbol" terms $[\mathbf{x}_i]$ and coefficients $[\alpha_i]$ respectively. Similarly, the components of $\sum_{i,j} \alpha_{ij} \mathbf{x}_i \mathbf{x}_j$ are known as the "double-symbol" terms and coefficients; and the components of $\sum_{i,j,k} \alpha_{ijk} (\mathbf{x}_i \mathbf{x}_j) \mathbf{x}_k$ are the "triple-symbol" terms and coefficients.

Addition is performed elementwise among the single-, double-, and triple- components; the result is the (formal) composition of the three results. Given

$$A = \sum_i \alpha_i \mathbf{x}_i + \sum_{i,j} \alpha_{ij} \mathbf{x}_i \mathbf{x}_j + \sum_{i,j,k} \alpha_{ijk} (\mathbf{x}_i \mathbf{x}_j) \mathbf{x}_k$$

$$B = \sum_i \beta_i \mathbf{x}_i + \sum_{i,j} \beta_{ij} \mathbf{x}_i \mathbf{x}_j + \sum_{i,j,k} \beta_{ijk} (\mathbf{x}_i \mathbf{x}_j) \mathbf{x}_k$$

(where the sums run from 1 to n), we define the sum $A + B$ to be

$$\sum_i (\alpha_i + \beta_i)\mathbf{x}_i + \sum_{i,j} (\alpha_{ij} + \beta_{ij})\mathbf{x}_i\mathbf{x}_j + \sum_{i,j,k} (\alpha_{ijk} + \beta_{ijk})(\mathbf{x}_i\mathbf{x}_j)\mathbf{x}_k$$

Multiplication is slightly more involved. We define the product AB to be

$$\sum_{i,j} \alpha_i\beta_{ij}\mathbf{x}_i\mathbf{x}_j + \sum_{i,j,k} \alpha_{ij}\beta_k(\mathbf{x}_i\mathbf{x}_j)\mathbf{x}_k - \sum_{i,j,k} \alpha_i\beta_{jk}(\mathbf{x}_i\mathbf{x}_j)\mathbf{x}_k.$$

The minus sign in front of the third term embodies antiassociativity.

3. The package

The **evitaicossa** package implements these relations in the context of an R-centric suite of software. I give some examples of the package in use. A good place to start is function `raaa()`, which returns a simple random element of the free antiassociative algebra:

```
> raaa()

free antiassociative algebra element:
+1a +3b +2d +3a.b +1c.b +1c.c +1(a.b)a +1(b.b)c +1(b.c)a
```

(the default alphabet for this command is $\{\mathbf{a}, \mathbf{b}, \mathbf{c}, \mathbf{d}\}$). We see the print method for the package which shows some of the structure of the object. This one has some single-symbol elements, some double-symbol and some triple-symbol elements.

It is possible to create elements using the `aaa()` or `as.aaa()` functions:

```
> x  <- as.aaa(c("p","q","r"))
> x1 <- aaa(s1 = c("p","r","x"),c(-1,5,6))
> y <- aaa(d1 = letters[1:3],d2 = c("foo","bar","baz"),dc=1:3)
> z <- aaa(
+         t1 = c("bar","bar","bar"),
+         t2 = c("q","r","s"),
+         t3 = c("foo","foo","bar"),
+         tc = 5:7)
```

And then apply arithmetic operations to these objects:

```
> x

free antiassociative algebra element:
+1p +1q +1r

> x1
```

```
free antiassociative algebra element:
-1p +5r +6x
```

```
> x+x1
```

```
free antiassociative algebra element:
+1q +6r +6x
```

(above, note the cancellation in `x+x1`). Multiplication is also implemented (package idiom is to use an asterisk):

```
> x*(x1+y)
```

```
free antiassociative algebra element:
-1p.p +5p.r +6p.x -1q.p +5q.r +6q.x -1r.p +5r.r +6r.x -1(p.a)foo -2(p.b)bar
-3(p.c)baz -1(q.a)foo -2(q.b)bar -3(q.c)baz -1(r.a)foo -2(r.b)bar -3(r.c)baz
```

Check:

```
> x*(x1+y) == x*x1 + x*y
```

```
[1] TRUE
```

We end with a remarkable identity:

$$(\mathbf{a} + \mathbf{ax})(\mathbf{b} + \mathbf{xb}) = \mathbf{ab}$$

Numerically:

```
> a <- raaa()
> b <- raaa()
> x <- raaa()
> (a+a*x)*(b+x*b) == a*b
```

```
[1] TRUE
```

4. Extract and replace methods

Because of the tripartite nature of antiassociative algebra, the package provides three families of extraction methods: `single()`, `double()` and `triple()`, which return the different components of an object:

```
> a
```

```
free antiassociative algebra element:
+4a +2b +2a.a +2c.c +2d.d +2(b.d)d +1(c.d)a +4(d.b)c

> single(a)

free antiassociative algebra element:
+4a +2b

> double(a)

free antiassociative algebra element:
+2a.a +2c.c +2d.d

> triple(a)

free antiassociative algebra element:
+2(b.d)d +1(c.d)a +4(d.b)c
```

The corresponding replacement methods are also implemented:

```
> single(a) <- 0
> a

free antiassociative algebra element:
+2a.a +2c.c +2d.d +2(b.d)d +1(c.d)a +4(d.b)c

> double(a) <- double(b) * 1000
> a

free antiassociative algebra element:
+1000b.d +2000c.b +2000c.d +2(b.d)d +1(c.d)a +4(d.b)c
```

Square bracket extraction and replacement is also implemented:

```
> (a <- raaa(s=5))

free antiassociative algebra element:
+4a +3b +6c +3b.c +1b.d +2c.c +3d.a +2d.c +2(a.d)d +3(b.a)a +4(b.b)c +1(c.d)d
+3(d.c)b

> a[s1=c("c","e"),t1="c",t2="d",t3="d"]

free antiassociative algebra element:
+6c +1(c.d)d
```

Above we pass named arguments (`s1` *et seq.*) and the appropriate `aaa` object is returned. Zero coefficients are discarded. This mode also implements replacement methods:

```
> (a <- raaa(s=5))

free antiassociative algebra element:
+8a +2b +2c +1a.b +4a.c +3b.c +4c.c +2(a.c)c +4(b.c)a +4(b.c)d +1(c.d)d
+3(d.c)a

> a[s1="a",d1=c("c","w"),d2=c("d","w")] <- 888
> a

free antiassociative algebra element:
+888a +2b +2c +1a.b +4a.c +3b.c +4c.c +888c.d +888w.w +2(a.c)c +4(b.c)a
+4(b.c)d +1(c.d)d +3(d.c)a
```

The other square bracket method is to pass an (unnamed) character vector:

```
> (a <- raaa())

free antiassociative algebra element:
+1a +2c +4d +3b.a +3b.b +3b.c +2(b.c)a +4(c.a)d +4(c.d)b
```

5. Note on disordR discipline

If we try to access the symbols or coefficients of an `aaa` object [functions `s1()` and `sc()` respectively], we get a `disord` object (Hankin 2022b). Suppose we wish to extract the single-symbol terms and the single-symbol coefficients:

```
> x

free antiassociative algebra element:
+3b +4c +1d +3a.c +2a.d +4d.a +1(a.a)d +1(a.b)d +3(d.d)a

> s1(x)

A disord object with hash b3ead18cdb285c1bd26ea0d9faf1dfcff2debd81 and elements
[1] "b" "c" "d"
(in some order)

> sc(x)

A disord object with hash b3ead18cdb285c1bd26ea0d9faf1dfcff2debd81 and elements
[1] 3 4 1
(in some order)
```

See how the hash codes of the symbols and coefficients match. However, the double-symbol terms and coefficients, while internally matching, differ from the single-symbol stuff:

```
> list(d1(x),d2(x),dc(x))

[[1]]
A disord object with hash eeb24e0f65a9b27e547fc0c7be14e7b33be06cd0 and elements
[1] "a" "a" "d"
(in some order)

[[2]]
A disord object with hash eeb24e0f65a9b27e547fc0c7be14e7b33be06cd0 and elements
[1] "c" "d" "a"
(in some order)

[[3]]
A disord object with hash eeb24e0f65a9b27e547fc0c7be14e7b33be06cd0 and elements
[1] 3 2 4
(in some order)
```

Above, see how the double-symbol terms and double-symbol coefficients have consistent hashes, but do not match the single-symbol objects (or indeed the triple-symbol objects).

5.1. Matrix index extraction

If square bracket extraction is given an index that is a matrix, it is interpreted rowwise:

```
> l <- letters[1:3]
> (a <- aaa(s1=l,sc=1:3, d1=l,d2=rev(l),dc=3:1,t1=l,t2=l,t3=rev(l),tc=1:3))

free antiassociative algebra element:
+1a +2b +3c +3a.c +2b.b +1c.a +1(a.a)c +2(b.b)b +3(c.c)a

> a[cbind(l,l)]

free antiassociative algebra element:
+2b.b

> a[cbind(rev(l),l,l)] <- 88
> a

free antiassociative algebra element:
+1a +2b +3c +3a.c +2b.b +1c.a +1(a.a)c +88(a.c)c +88(b.b)b +88(c.a)a +3(c.c)a
```

6. Note on generalized antiassociativity

We may generalize antiassociativity to $\mathbf{a}(\mathbf{bc}) = k(\mathbf{ab})\mathbf{c}$. Thus associativity is recovered if $k = 1$ and antiassociativity if $k = -1$. Then the nilpotence argument becomes:

$$(\mathbf{ab})(\mathbf{cd}) = k^{-1}\mathbf{a}(\mathbf{b}(\mathbf{cd})) = \mathbf{a}((\mathbf{bc})\mathbf{d}) = k(\mathbf{a}(\mathbf{bc}))\mathbf{d} = k^2((\mathbf{ab})\mathbf{c})\mathbf{d} = k(\mathbf{ab})(\mathbf{cd})$$

The value of k may be set at compile-time by editing file `src/anti.h`. The line in question reads:

```
#define K -1 // a(bc) == K(ab)c
```

but it is possible to change the value of `K`. Note that this will cause `test_aac.R`, one of the `testthat` suite, to fail `R CMD check`.

As noted above, Remm (2024) uses $\mathbf{x}_i(\mathbf{x}_j\mathbf{x}_k)$ rather than $(\mathbf{x}_i\mathbf{x}_j)\mathbf{x}_k$ for the triple products. I chose the latter because R idiom for multiplication is left associative:

```
> x <- 3
> class(x) <- "foo"
> `*.foo` <- function(x,y){x + y + x}
> print.foo <- function(x){print(unclass(x))}
> c(`(x*x)*x` = (x*x)*x,  `x*(x*x)` = x*(x*x),  `x*x*x` = x*x*x)

(x*x)*x x*(x*x)   x*x*x
     21      15      21
```

Above we see that `x*x*x` is interpreted as `(x*x)*x`, which is why the sign convention in the package was adopted.

References

Hankin RKS (2006). "Normed division algebras with R: introducing the onion package." *R News*, **6**(2), 49–52.

Hankin RKS (2022a). "Clifford algebra in R." https://arxiv.org/abs/2209.13659. doi:10.48550/ARXIV.2209.13659.

Hankin RKS (2022b). "Disordered vectors in R: introducing the **disordR** package." https://arxiv.org/abs/2210.03856. doi:10.48550/ARXIV.2210.03856.

Hankin RKS (2022c). "Fast multivariate polynomials in R: the **mvp** package." https://arxiv.org/abs/2210.15991.

Hankin RKS (2022d). "The free algebra in R." https://arxiv.org/abs/2211.04002. doi:10.48550/ARXIV.2211.04002.

Hankin RKS (2022e). "Quantum algebra in R: the **weyl** package." https://arxiv.org/abs/2212.09230. doi:10.48550/ARXIV.2212.09230.

Hankin RKS (2022f). "Sparse arrays in R: the **spray** package." https://arxiv.org/abs/2210.03856. doi:10.48550/ARXIV.2210.10848. URL https://arxiv.org/abs/2210.10848.

Hankin RKS (2023). "Jordan algebra in R." https://arxiv.org/2303.06062. doi:10.48550/arXiv.2303.06062.

Hankin RKS (2024). "Antiassociative algebra in R: introducing the **evitaicossa** package." doi:10.48550/arXiv.2412.16161.

Remm E (2024). "Anti-associative algebras." https://arxiv.org/abs/2202.10812. 2202.10812.

Affiliation:

Robin K. S. Hankin
University of Stirling
E-mail: hankin.robin@gmail.com

Submitted: 0000-00-00
Accepted: 0000-00-00