The `freealg` package: the free algebra in R

Robin K. S. Hankin

2019-09-16

Introduction

The `freealg` package provides some functionality for the free algebra, using the Standard Template library of `C++`, commonly known as the `STL`. It is very like the `mvp` package for multivariate polynomials, but the indeterminates do not commute: multiplication is word concatenation.

The free algebra

A vector space over a commutative field $\mathbb{F}$ (here the reals) is a set $\mathbb{V}$ together with two binary operations, addition and scalar multiplication. Addition, usually written $+$, makes $(\mathbb{V},+)$ an Abelian group. Scalar multiplication is usually denoted by juxtaposition (or sometimes $\times$) and makes $(\mathbb{V},\cdot)$ a semigroup. In addition, for any $a,b\in\mathbb{F}$ and $\mathbf{u},\mathbf{v}\in\mathbb{V}$, the following laws are satisfied:

- Compatibility: $a\left(b\mathbf{v}\right)=(ab)\mathbf{v}$
- Identity: $1\mathbf{v}=\mathbf{v}$, where $1\in\mathbb{F}$ is the multiplicative identity
- Distributivity of vector addition: $a\left(\mathbf{u}+\mathbf{v}\right)=a\mathbf{u}+b\mathbf{v}$
- Distributivity of field addition: $(a+b)\mathbf{v}=a\mathbf{v}+b\mathbf{v}$

An *algebra* is a vector space endowed with a binary operation, usually denoted by either a dot or juxtaposition of vectors, from $\mathbb{V}\times\mathbb{V}$ to $\mathbb{V}$ satisfying:

- Left distributivity: $\mathbf{u}\cdot(\mathbf{v}+\mathbf{w})=\mathbf{u}\cdot\mathbf{v}+\mathbf{u}\cdot\mathbf{w}$.
- Right distributivity: $(\mathbf{u}+\mathbf{v})\cdot\mathbf{w}=\mathbf{u}\cdot\mathbf{w}+\mathbf{v}\cdot\mathbf{w}$.
- Compatibility: $(a\mathbf{u})\cdot(b\mathbf{v})=(ab)(\mathbf{u}\cdot\mathbf{v})$.

There is no requirement for vector multiplication to be commutative or indeed associative.

The *free algebra* is, in one sense, the most general algebra. Given a set of indeterminates $\{X_1,X_2,\ldots,X_n\}$ one considers the set of *words*, that is, finite sequences of indeterminates. The vector space is then scalar multiples of words together with a formal vector addition. This system automatically satisfies the axioms of an algebra together with the requirement that addition of vectors satisfies distributivity.

The free algebra is the free R-module with a basis consisting of all words over an alphabet of symbols with multiplication of words defined as concatenation. Thus, with an alphabet of $\{x,y,z\}$ and

$$A=\alpha x^2yx+\beta zy\qquad B=\gamma z+\delta y^4$$

we would have

$$A\cdot B=\left(\alpha x^2yx+\beta zy\right)\cdot\left(\gamma z+\delta y^4\right)=\alpha\gamma x^2yxz+\alpha\delta x^2yxy^4+\beta\gamma zyz+\beta\delta zy^5$$

and

$$B\cdot A=\left(\gamma z+\delta y^4\right)\cdot\left(\alpha x^2yx+\beta zy\right)=\alpha\gamma zx^2yx+\beta\gamma z^2y+\alpha\delta y^4x^2yx+\beta\delta y^4zy.$$

Note that multiplication is not commutative, but it is associative.

The STL map class

Here, a *term* is defined to be a scalar multiple of a *word*, and an element of the free algebra is considered to be the sum of a finite number of *terms*.

Thus we can view an element of the free algebra as a map from the set of words to the reals, each word mapping to its coefficient. Note that the empty word maps to the constant term. In STL terminology, a `map` is a sorted associative container that contains key-value pairs with unique keys. It is used in the package to map words to their coefficients, and is useful here because search and insertion operations have logarithmic complexity.

Package conventions

The indeterminates are the strictly positive integers (that is, we identify X_i with integer i); a natural and easily implemented extension is to allow each indeterminate X_i to have an inverse X_{-i}.

It is natural to denote indeterminates $X_1, X_2, X_3 \ldots$ with lower-case letters $a, b, c, \ldots$ and their multiplicative inverses with upper-case letters $A, B, C \ldots$

Thus we might consider $X = 5a + 43xy + 6yx - 17a^3b$ to be the map

```
{[1] -> 5, [24,25] -> 43, [25,24] -> 6, [1,1,1,2] -> -17}
```

In standard template library language, this is a map from a list of signed integers to a double; the header in the package reads

```
typedef std::list<signed int> word; // an 'word' object is a list of signed ints
typedef map <word, double> freealg; // a 'freealg' maps word objects to reals
```

Although there are a number of convenience wrappers in the package, we would create object X as follows:

```
library("freealg")
X <- freealg(words = list(1, c(24,25), c(25,24), c(1,1,1,2)), coeffs = c(5, 43, 6, -17))
dput(X)
# structure(list(indices = list(1L, c(1L, 1L, 1L, 2L), 24:25, 25:24),
#     coeffs = c(5, -17, 43, 6)), class = "freealg")
X
# free algebra element algebraically equal to
#  + 5*a - 17*aaab + 43*xy + 6*yx
```

(the print method translates from integers to letters). Note that the key-value pairs do not have a well-defined order in the `map` class; so the terms of a free algebra object may appear in any order. This does not affect the algebraic value of the object and allows for more efficient storage and manipulation. See also the `mvp` (Hankin 2019a) and `spray` (Hankin 2019b) packages for similar issues.

The package in use

There are a variety of ways of creating free algebra objects:

```
(X <- as.freealg("3aab -2abbax"))  # caret ("^") not yet implemented
# free algebra element algebraically equal to
#  + 3*aab - 2*abbax
(Y <- as.freealg("2 -3aab -aBBAA"))  # uppercase letters are inverses
# free algebra element algebraically equal to
#  + 2 - 1*aBBAA - 3*aab
```

```
(Z <- as.freealg(1:3))
# free algebra element algebraically equal to
#  + 1*a + 1*b + 1*c
```

Then the usual arithmetic operations work, for example:

```
X^2        # powers are implemented
# free algebra element algebraically equal to
#  + 9*aabaab - 6*aababbax - 6*abbaxaab + 4*abbaxabbax
X+Y        #  'aab' term cancels
# free algebra element algebraically equal to
#  + 2 - 1*aBBAA - 2*abbax
1000+Y*Z   # algebra multiplication and addition works as expected
# free algebra element algebraically equal to
#  + 1000 + 2*a - 1*aBBA - 1*aBBAAb - 1*aBBAAc - 3*aaba - 3*aabb - 3*aabc + 2*b + 2*c
```

The print method

The default print method uses uppercase letters to represent multiplicative inverses, but it is possible to set the `usecaret` option, which makes changes the appearance:

```
set.seed(0)
phi <- rfalg(n=5,inc=TRUE)
phi
# free algebra element algebraically equal to
#  + 4*B + 3*BC + 1*a + 5*aCBB + 2*bc
options("usecaret" = TRUE)
phi
# free algebra element algebraically equal to
#  + 4*b^-1 + 3*b^-1c^-1 + 1*a + 5*ac^-1b^-1b^-1 + 2*bc
options("usecaret" = FALSE)  # reset to default
```

An example

Haiman (1993) poses the following question, attributed to David Richman:

"find the constant term of $(x + y + x^{-1} + y^{-1})^p$ when x and y do not commute".

Package idiom is straightforward. Consider $p = 4$:

```
X <- as.freealg("x+y+X+Y")
X^2
# free algebra element algebraically equal to
#  + 4 + 1*YY + 1*YX + 1*Yx + 1*XY + 1*XX + 1*Xy + 1*xY + 1*xx + 1*xy + 1*yX + 1*yx + 1*yy
constant(X^4)
# [1] 28
```

We could even calculate the first few terms of Sloane's A035610:

```
f <- function(n){constant(as.freealg("x+y+X+Y")^n)}
sapply(c(0,2,4,6,8,10),f)
# [1]     1     4    28   232  2092 19864
```

References

Haiman, M. 1993. "Non-Commutative Rational Power Series and Algebraic Generating Functions." *European Journal of Combinatorics* 14 (4): 335–39.

Hankin, Robin K. S. 2019a. *mvp: Fast Symbolic Multivariate Polynomials*. https://github.com/RobinHankin/mvp.git.

———. 2019b. *Spray: Sparse Arrays and Multivariate Polynomials*. https://github.com/RobinHankin/spray.git.