Multivariate polynomials in R

Robin K. S. Hankin
Auckland University of Technology

Abstract

In this short article I introduce the **multipol** package, which provides some functionality for handling multivariate polynomials; the package is discussed here from a programming perspective. An example from the field of enumerative combinatorics is presented.

The package is almost completely superceded by the **mvp** and **spray** packages (Hankin 2022b,c) which use a sparse array technique and follow **disordR** discipline (Hankin 2022a). This vignette is based on Hankin (2008); the discussion of sparsity is unchanged from 2008.

Keywords: Multivariate polynomials, R.

1. Univariate polynomials

A *polynomial* is an algebraic expression of the form $\sum_{i=0}^{n} a_i x^i$ where the a_i are real or complex numbers and n (the *degree* of the polynomial) is a nonnegative integer. A polynomial may be viewed in three distinct ways:

- Polynomials are interesting and instructive examples of complete functions: they map $\mathbb{C}$ (the complex numbers) to $\mathbb{C}$.
- Polynomials are a map from the positive integers to $\mathbb{C}$: this is $f(n) = a_n$ and one demands that $\exists n_0$ with $n \geqslant n_0 \longrightarrow f(n) = 0$. Relaxation of the final clause results in a *generating function* which is useful in combinatorics.
- Polynomials with complex coefficients form an algebraic object known as a *ring*: polynomial multiplication is associative and distributive with respect to addition; $(ab)c = a(bc)$ and $a(b + c) = ab + ac$.

A *multivariate polynomial* is a generalization of a polynomial to expressions of the form $\sum a_{i_1 i_2 \ldots i_d} \prod_{j=1}^{d} x_j^{i_j}$. The three characterizations of polynomials above generalize to the multivariate case, but note that the algebraic structure is more general.

In the context of R programming, the first two points typically dominate. Viewing a polynomial as a function is certainly second nature to the current readership and unlikely to yield new insight. But generating functions are also interesting and useful applications of polynomials (Wilf 1994) which may be less familiar and here I discuss an example from the discipline of integer partitions (Andrews 1998).

A *partition of an integer* n is a non-increasing sequence of positive integers $p_1, p_2, \ldots, p_r$ such that $n = \sum_{i=1}^{r} p_i$ (Hankin 2006a). How many distinct partitions does n have?

The answer is the coefficient of x^n in

$$\prod_{i=1}^{n} \frac{1}{1-x^i}$$

(observe that we may truncate the Taylor expansion of $1/(1-x^j)$ to terms not exceeding x^n; thus the problem *is* within the domain of polynomials as infinite sequences of coefficients are not required). Here, as in many applications of generating functions, one uses the mechanism of polynomial multiplication as a bookkeeping device to keep track of the possibilities. The R idiom used in the **polynom** package is a spectacularly efficient method for doing so.

Multivariate polynomials generalize the concept of generating function, but in this case the functions are from n-tuples of nonnegative integers to $\mathbb{C}$. An example is given in the appendix below.

1.1. The polynom package

The **polynom** package (Venables, Hornik, and Maechler 2007) is a consistent and convenient suite of software for manipulating polynomials. This package was originally written in 1993 and is used by Venables and Ripley (2001) as an example of S3 classes.

The following R code shows the **polynom** package in use; the examples are then generalized to the multivariate case using the **multipol** package.

```
> require(polynom)
> (p <- polynomial(c(1,0,0,3,4)))

1 + 3*x^3 + 4*x^4

> str(p)

 'polynomial' num [1:5] 1 0 0 3 4
```

See how a polynomial is represented as a vector of coefficients with `p[i]` holding the coefficient of x^{i-1}; note the off-by-one issue. Observe the natural print method which suppresses the zero entries—but the internal representation requires all coefficients so a length 5 vector is needed to store the object.

Polynomials may be multiplied and added:

```
> p + polynomial(1:2)

2 + 2*x + 3*x^3 + 4*x^4

> p*p

1 + 6*x^3 + 8*x^4 + 9*x^6 + 24*x^7 + 16*x^8
```

Note the overloading of '+' and '*': polynomial addition and multiplication are executed using the natural syntax on the command line. Observe that the addition is not entirely straightforward: the shorter polynomial must be padded with zeroes.

A polynomial may be viewed either as an object, or a function. Coercing a polynomial to a function is straightforward:

```
> f1 <- as.function(p)
> f1(pi)

[1] 483.6552

> f1(matrix(1:6,2,3))

     [,1] [,2] [,3]
[1,]    8  406 2876
[2,]   89 1217 5833
```

Note the effortless and transparent vectorization of `f1()`.

2. Multivariate polynomials

There exist several methods by which polynomials may be generalized to multipols. To this author, the most natural is to consider an array of coefficients; the dimensionality of the array corresponds to the arity of the multipol. However, other methods suggest themselves and a brief discussion is given at the end.

Much of the univariate polynomial functionality presented above is directly applicable to multivariate polynomials.

```
> require(multipol)
> (a <- as.multipol(matrix(1:10,nrow=2)))

    y^0 y^1 y^2 y^3 y^4
x^0   1   3   5   7   9
x^1   2   4   6   8  10
```

See how a multipol is actually an array, with one extent per variable present, in this case 2, although the package is capable of manipulating polynomials of arbitrary arity.

Multipol addition is a slight generalization of the univariate case:

```
> b <-  as.multipol(matrix(1:10,ncol=2))
> a+b

    y^0 y^1 y^2 y^3 y^4
x^0   2   9   5   7   9
x^1   4  11   6   8  10
x^2   3   8   0   0   0
x^3   4   9   0   0   0
x^4   5  10   0   0   0
```

In the multivariate case, the zero padding must be done in each array extent; the natural command-line syntax is achieved by defining an appropriate `Ops.multipol()` function to overload the arithmetic operators.

2.1. Multivariate polynomial multiplication

The heart of the package is multipol multiplication:

```
> a * b
```

```
    y^0 y^1 y^2 y^3 y^4 y^5
x^0   1   9  23  37  51  54
x^1   4  29  61  93 125 123
x^2   7  39  79 119 159 142
x^3  10  49  97 145 193 161
x^4  13  59 115 171 227 180
x^5  10  40  70 100 130 100
```

Multivariate polynomial multiplication is considerably more involved than in the univariate case. Consider the coefficient of x^2y^2 in the product. This is

$$\begin{aligned}
& C_a\left(x^2y^2\right)C_b\left(1\right) + C_a\left(xy^2\right)C_b\left(x\right) + C_a\left(y^2\right)C_b\left(x^2\right) \\
& + C_a\left(x^2y\right)C_b\left(y\right) + C_a\left(xy\right)C_b\left(xy\right) + C_a\left(y\right)C_b\left(x^2y\right) \\
& + C_a\left(x^2\right)C_b\left(y^2\right) + C_a\left(x\right)C_b\left(xy^2\right) + C_a\left(1\right)C_b\left(x^2y^2\right) \\
= \;& 0\cdot 1 + 6\cdot 2 + 5\cdot 3 \\
& + 0\cdot 6 + 4\cdot 7 + 3\cdot 8 \\
& + 0\cdot 0 + 2\cdot 0 + 1\cdot 0 \\
= \;& 79,
\end{aligned}$$

where "$C_a\left(x^my^n\right)$" means the coefficient of x^my^n in polynomial `a`. It should be clear that large multipols involve more terms and a typical example is given later in the paper.

Multivariate polynomial multiplication in ***multipol***

The appropriate R idiom is to follow the above prose description in a vectorized manner; the following extract from `mprod()` is very slightly edited in the interests of clarity.

First we define a matrix, `index`, whose rows are the array indices of the product:

```
outDims <- dim(a)+dim(b)-1
```

Here `outDims` *is the dimensions of the product. Note again the off-by-one issue: the package uses array indices internally, while the user consistently indexes by variable power.*

```
index <- expand.grid(lapply(outDims,seq_len))
```

Each row of matrix `index` *is thus an array index for the product.*

The next step is to define a convenience function `f()`*, whose argument* `u` *is a row of* `index`*, that returns the entry in the multipol product:*

```
f <- function(u){
  jja <-
  expand.grid(lapply(u,function(i)0:(i-1)))
  jjb <- -sweep(jja, 2, u)-1
```

So `jja` *is the (power) index of* `a`*, and the rows of* `jjb` *added to those of* `jja` *give* `u`*, which is the power index of the returned array. Now not all rows of* `jja` *and* `jjb` *correspond to extant elements of* `a` *and* `b` *respectively; so define a Boolean variable* `wanted` *that selects just the appropriate rows:*

```
wanted <-
apply(jja,1,function(x)all(x < dim(a))) &
apply(jjb,1,function(x)all(x < dim(b))) &
apply(jjb,1,function(x)all(x >= 0))
```

Thus element `n` *of* `wanted` *is* `TRUE` *only if the* `n`*th row of both* `jja` *and* `jjb` *correspond to a legal element of* `a` *and* `b` *respectively. Now perform the addition by summing the products of the legal elements:*

```
sum(a[1+jja[wanted,]] * b[1+jjb[wanted,]])
}
```

Thus function `f()` *returns the coefficient, which is the sum of products of pairs of legal elements of* `a` *and* `b`*. Again observe the off-by-one issue.*

Now `apply()` *function* `f()` *to the rows of* `index` *and reshape:*

```
out <- apply(index,1,f)
dim(out) <- outDims
```

Thus array `out` *contains the multivariate polynomial product of* `a` *and* `b`*.*

The preceding code shows how multivariate polynomials may be multiplied. The implementation makes no assumptions about the entries of `a` or `b` and the coefficients of the product are summed over all possibilities; opportunities to streamline the procedure are discussed below.

2.2. Multipols as functions

Polynomials are implicitly functions of one variable; multivariate polynomials are functions too, but of more than one argument. Coercion of a multipol to a function is straightforward:

```
> f2 <- as.function(a*b)

> f2(c(x=1,y=3i))

[1] 67725+167400i
```

It is worth noting the seamless integration between **polynom** and **multipol** in this regard: `f1(a)` is a multipol [recall that `f1()` is a function coerced from a univariate polynomial].

2.3. Multipol extraction and replacement

One often needs to extract or replace parts of a multipol. The package includes extraction and replacement methods but, partly because of the off-by-one issue, these are not straightforward.

Consider the case where one has a multipol and wishes to extract the terms of order zero ane one:

```
> a[0:1,0:1]

     [,1] [,2]
[1,]    1    3
[2,]    2    4
```

Note how the off-by-one issue is handled: `a[i,j]` is the coefficient of $x^i y^j$ (here the constant and first-order terms); the code is due to Rougier (2007). Replacement is slightly different:

```
> a[0,0] <- -99
> a

    y^0 y^1 y^2 y^3 y^4
x^0 -99   3   5   7   9
x^1   2   4   6   8  10
```

Observe how replacement operators—unlike extraction operators—return a multipol; this allows expeditious modification of multivariate polynomials. The reason that the extraction operator returns an array rather than a multipol is that the extracted object often does not have unambiguous interpretation as a multipol (consider `a[-1,-1]`, for example). It seems to this author that the loss of elegance arising from the asymmetry between extraction and replacement is amply offset by the impossibility of an extracted object's representation as a multipol being undesired—unless the user explicitly coerces.

3. The elephant in the room

Representing a multivariate polynomial by an array is a natural and efficient method, but suffers some disadvantages.

Consider Euler's four-square identity

$$\begin{aligned}\left(a_1^2+a_2^2+a_3^2+a_4^2\right)\cdot\left(b_1^2+b_2^2+b_3^2+b_4^2\right) = \\ \left(a_1b_1-a_2b_2-a_3b_3-a_4b_4\right)^2+ \\ \left(a_1b_2+a_2b_1+a_3b_4-a_4b_3\right)^2+ \\ \left(a_1b_3-a_2b_4+a_3b_1+a_4b_2\right)^2+ \\ \left(a_1b_4+a_2b_3-a_3b_2+a_4b_1\right)^2\end{aligned}$$

which was discussed in 1749 in a letter from Euler to Goldbach. The identity is important in number theory, and may be proved straightforwardly by direct expansion[1]. It may by verified to machine precision using the **multipol** package; the left hand side is given by:

```
> options("showchars" = TRUE)
> lhs <- polyprod(ones(4,2),ones(4,2))

[1] "1*x1^2*x5^2 + 1*x2^2*x5^2 + ...
```

(the right hand side's idiom is more involved), but this relatively trivial expansion requires about 20 minutes on my 1.5 GHz G4; the product comprises $3^8 = 6561$ elements, of which only 16 are nonzero. Note the `options()` statement controlling the format of the output which causes the result to be printed in a more appropriate form. Clearly the **multipol** package as currently implemented is inefficient for multivariate problems of this nature in which the arrays possess few nonzero elements.

A challenge

The inefficiency discussed above is ultimately due to the storage and manipulation of many zero coefficients that may be omitted from a calculation. Multivariate polynomials for which this is an issue appear to be common: the package includes many functions—such as `uni()`, `single()`, and `lone()`—that define useful multipols in which the number of nonzero elements is very small.

In this section, I discuss some ideas for implementations in which zero operations are implicitly excluded. These ideas are presented in the spirit of a request for comments: although they seem to this author to be reasonable methodologies, readers are invited to discuss the ideas presented here and indeed to suggest alternative strategies.

The canonical solution would be to employ some form of sparse array class, along the lines of Mathematica's `SparseArray`. Unfortunately, no such functionality exists as of 2008, but C++ includes a "map" class (Stroustrup 1997) that would be ideally suited to this application.

There are other paradigms that may be worth exploring. It is possible to consider a multivariate polynomial of arity d (call this an object of class P^d) as being a univariate polynomial whose coefficients are of class P^{d-1}—class P^0 would be a real or complex number—but such recursive class definitions appear not to be possible with the current implementation of S3 or S4 (Venables 2008). Recent experimental work by West (2008) exhibits a proof-of-concept in C++ which might form the back end of an R implementation. Euler's identity appears to be a particularly favourable example and is proved essentially instantaneously.

4. Conclusions

This short document introduces the **multipol** package that provides functionality for manipulating multivariate polynomials. The **multipol** package builds on and generalizes the **polynom**

[1] Or indeed more elegantly by observing that both sides of the identity express the absolute value of the product of two quaternions: $|a|^2\,|b|^2 = |ab|^2$. With the **onion** package (Hankin 2006b), one would define `f <- function(a,b)Norm(a)*Norm(b) - Norm(a*b)` and observe (for example) that `f(rquat(rand="norm"),rquat(rand="norm"))` is zero to machine precision.

package of Venables *et al.*, which is restricted to the case of univariate polynomials. The generalization is not straightforward and presents a number of programming issues that were discussed.

One overriding issue is that of performance: many multivariate polynomials of interest are "sparse" in the sense that they have many zero entries that unnecessarily consume storage and processing resources.

Several possible solutions are suggested, in the form of a request for comments. The canonical method appears to be some form of sparse array, for which the "map" class of the C++ language is ideally suited. Implementation of such functionality in R might well find application in fields other than multivariate polynomials.

5. An example

This appendix presents a brief technical example of multivariate polynomials in use in the field of enumerative combinatorics (Good 1976). Suppose one wishes to determine how many contingency tables, with non-negative integer entries, have specified row and column marginal totals. The appropriate generating function is

$$\prod_{1\leqslant i\leqslant nr}\prod_{1\leqslant j\leqslant nc}\frac{1}{1-x_i y_j}$$

where the table has nr rows and nc columns (the number of contingency tables is given by the coefficient of $x_1^{s_1}x_2^{s_2}\cdots x_r^{s_r}\cdot y_1^{t_1}y_2^{t_2}\cdots y_t^{t_c}$ where the s_i and t_i are the row- and column- sums respectively). The R idiom for the generating function `gf` in the case of $nr = nc = n = 3$ is:

```
 n <- 3
jj <- as.matrix(expand.grid(seq_len(n),n+seq_len(n)))
 f <- function(i) ooom(n,lone(2*n,jj[i,]),maxorder=n)
 u <- c(sapply(seq_len(n^2),f,simplify=FALSE))
gf <- do.call("mprod", u)
```

[here function `ooom()` is "one-over-one-minus"; and `mprod()` is the function name for multipol product]. In this case, it is clear that sparse array functionality would not result in better performance, as almost every element of the generating function `gf` is nonzero. Observe that the maximum of `gf`, 55, is consistent with Sloane (2008).

Acknowledgements

I would like to acknowledge the many stimulating comments made by the R-help list. In particular, the insightful comments from Bill Venables and Kurt Hornik were extremely helpful.

References

Andrews GE (1998). *The Theory of Partitions*. Cambridge University Press.

Euler L (1749). "Lettre CXXV." Communication to Goldbach; Berlin, 12 April.

Good IJ (1976). "On the application of symmetric Dirichlet distributions and their mixtures to contingency tables." *The Annals of Statistics*, **4**(6), 1159–1189.

Hankin RKS (2006a). "Additive Integer Partitions in R." *Journal of Statistical Software, Code Snippets*, **16**(1).

Hankin RKS (2006b). "Normed division algebras with R: Introducing the **onion** package." *R News*, **6**(2), 49–52. URL http://CRAN.R-project.org/doc/Rnews/.

Hankin RKS (2008). "Programmers' Niche: Multivariate polynomials in R." *R News*, **8**(1), 41–45. URL http://CRAN.R-project.org/doc/Rnews/.

Hankin RKS (2022a). "Disordered vectors in R: introducing the **disordR** package." doi:10.48550/ARXIV.2210.03856.

Hankin RKS (2022b). "Fast multivariate polynomials in R: the **mvp** package." doi:10.48550/ARXIV.2210.15991.

Hankin RKS (2022c). "Sparse arrays in R: the **spray** package." doi:10.48550/ARXIV.2210.10848.

Rougier J (2007). ***Oarray**: Arrays with arbitrary offsets*. R package version 1.4-2.

Sloane NJA (2008). "The On-Line Encyclopedia of Integer Sequences." Published electronically at http://www.research.att.com/~njas/sequences/A110058.

Stroustrup B (1997). *The C++ Programming Language*. Third edition. Addison Wesley.

Venables B, Hornik K, Maechler M (2007). ***polynom**: A collection of functions to implement a class for univariate polynomial manipulations*. R package version 1.3-2. S original by Bill Venables, packages for R by Kurt Hornik and Martin Maechler, URL http://CRAN.R-project.org/.

Venables WN (2008). Personal Communication.

Venables WN, Ripley BD (2001). *S Programming*. Springer.

West LJ (2008). "An experimental C++ implementation of a recursively defined polynomial class." Personal communication.

Wilf HS (1994). *generatingfunctionology*. Academic Press.

Affiliation:

Robin K. S. Hankin
Auckland University of Technology
New Zealand