The mvp package: fast multivariate polynomials R

Robin K. S. Hankin

2019-06-28

Introduction

The `mvp` package provides some functionality for fast manipulation of multivariate polynomials, using the Standard Template library of `C++`, commonly known as the `STL`. It is comparable in speed to the `spray` package for sparse arrays, while retaining the symbolic capabilities of the `mpoly` package (Kahle 2013). The `mvp package` uses the excellent print and coercion methods of `mpoly`. The `mvp` package provides improved speed over `mpoly`, the ability to handle negative powers, and a more sophisticated substitution mechanism.

The `STL map` class

A `map` is a sorted associative container that contains key-value pairs with unique keys. It is interesting here because search and insertion operations have logarithmic complexity. Multivariate polynomials are considered to be the sum of a finite number of *terms*, each multiplied by a coefficient. A *term* is something like x^2y^3z. We may consider this term to be the map

```
{"x" -> 2, "y" -> 3, "z" -> 1}
```

where the map takes symbols to their (integer) power; it is understood that powers are nonzero. A `mvp` object is a map from terms to their coefficients; thus $7xy^2 - 3x^2yz^5$ would be

```
{{"x" -> 1, "y" -> 2} -> 7, {"x" -> 2, 'y" -> 1, "z" ->5} -> -3}
```

and we understand that coefficients are nonzero. In `C++` the declarations would be

```
typedef vector <signed int> mypowers;
typedef vector <string> mynames;

typedef map <string, signed int> term;
typedef map <term, double> mvp;
```

Thus a `term` maps a string to a (signed) integer, and a `mvp` maps terms to doubles. One reason why the `map` class is fast is that the order in which the keys are stored is undefined: the compiler may store them in the order which it regards as most propitious. This is not an issue for the maps considered here as addition and multiplication are commutative and associative.

Note also that constant terms are handled with no difficulty (constants are simply maps from the empty map to its value), as is the zero polynomial (which is simply an empty map).

The package in use

Consider a simple multivariate polynomial $3xy + z^3 + xy^6z$ and its representation in the following R session:

```
library("mvp",quietly=TRUE)
(p <- as.mvp("3 x y + z^3 + x y^6 z"))
#> mvp object algebraically equal to
#> 3 x y  +  x y^6 z  +  z^3
```

Coercion and printing are accomplished by the `mpoly` package (there is no way I could improve upon Kahle's work). Note carefully that the printed representation of the mvp object is created by the `mpoly` package and the print method can rearrange both the terms of the polynomial ($3xy + z^3 + xy^6z = z^3 + 3xy + xy^6z$, for example) and the symbols within a term ($3xy = 3yx$, for example) to display the polynomial in a human-friendly form.

However, note carefully that such rearranging does not affect the mathematical properties of the polynomial itself. In the `mvp` package, the order of the terms is not preserved (or even defined) in the internal representation of the object; and neither is the order of the symbols within a single term. Although this might sound odd, if we consider a marginally more involved situation, such as

```
(M <- as.mvp("3 stoat goat^6 -4 + 7 stoatboat^3 bloat -9 float boat goat gloat^6"))
#> mvp object algebraically equal to
#> -4  +  7 bloat stoatboat^3  -  9 boat float gloat^6 goat  +  3 goat^6 stoat
dput(M)
#> structure(list(names = list(character(0), c("bloat", "stoatboat"
#> ), c("boat", "float", "gloat", "goat"), c("goat", "stoat")),
#>     power = list(integer(0), c(1L, 3L), c(1L, 1L, 6L, 1L), c(6L,
#>     1L)), coeffs = c(-4, 7, -9, 3)), class = "mvp")
```

it is not clear that any human-discernable ordering is preferable to any other, and we would be better off letting the compiler decide a propitious ordering. In any event, the `mpoly` package can specify a print order:

```
print(M,order="lex", varorder=c("stoat","goat","boat","bloat","gloat","float","stoatboat"))
#> mvp object algebraically equal to
#> 3 stoat goat^6  -  9 goat boat gloat^6 float  +  7 bloat stoatboat^3  -  4
```

Arithmetic operations

The arithmetic operations `*`, `+`, `-` and `^` work as expected:

```
(S1 <- rmvp(5,2,2,4))
#> mvp object algebraically equal to
#> 4 a^3  +  3 b^2 c  +  7 b^3  +  d^3
(S2 <- rmvp(5,2,2,4))
#> mvp object algebraically equal to
#> 2 a d  +  4 a^2 b  +  3 a^2 b^2  +  6 c^2 d
S1 + S2
#> mvp object algebraically equal to
#> 2 a d  +  4 a^2 b  +  3 a^2 b^2  +  4 a^3  +  3 b^2 c  +  7 b^3  +  6 c^2 d  +  d^3
S1 * S2
#> mvp object algebraically equal to
#> 6 a b^2 c d  +  14 a b^3 d  +  2 a d^4  +  4 a^2 b d^3  +  3 a^2 b^2 d^3  +  12 a^2 b^3 c  +  28 a^2
S1^2
#> mvp object algebraically equal to
#> 24 a^3 b^2 c  +  56 a^3 b^3  +  8 a^3 d^3  +  16 a^6  +  6 b^2 c d^3  +  14 b^3 d^3  +  9 b^4 c^2  +
```

Substitution

The package has two substitution functionalities. Firstly, we can substitute one or more variables for a numeric value. Define a mvp object:

```
(S3 <- as.mvp("x + 5 x^4 y + 8 y^2 x z^3"))
#> mvp object algebraically equal to
```

```
#> x  +  8 x y^2 z^3  +  5 x^4 y
```

And then we may substitute $x = 1$:

```
subs(S3, x = 1)
#> mvp object algebraically equal to
#> 1  +  5 y  +  8 y^2 z^3
```

Note the natural R idiom, and that the return value is another mvp object. We may subsitute for the other variables:

```
subs(S3, x = 1, y = 2, z = 3)
#> [1] 875
```

(in this case, the default behaviour is to return a "dropped" version of the resulting polynomial, that is, coerced to a scalar). The idiom also allows one to substitute a variable for an `mvp` object:

```
subs(as.mvp("a+b+c"), a="x^6")
#> mvp object algebraically equal to
#> b  +  c  +  x^6
```

Note carefully that `subs()` depends on the order of substitution:

```
subs(as.mvp("a+b+c"), a="x^6",x="1+a")
#> mvp object algebraically equal to
#> 1  +  6 a  +  15 a^2  +  20 a^3  +  15 a^4  +  6 a^5  +  a^6  +  b  +  c
subs(as.mvp("a+b+c"), x="1+a",a="x^6")
#> mvp object algebraically equal to
#> b  +  c  +  x^6
```

Pipes

Substitution works well with pipes:

```
as.mvp("a+b") %>% subs(a="a^2+b^2") %>% subs(b="x^6")
#> mvp object algebraically equal to
#> a^2  +  x^6  +  x^12
```

Differentiation

Differentiation is implemented. First we have the `deriv()` method:

```
(S <- as.mvp("a + 5 a^5*b^2*c^8 -3 x^2 a^3 b c^3"))
#> mvp object algebraically equal to
#> a  -  3 a^3 b c^3 x^2  +  5 a^5 b^2 c^8
deriv(S, letters[1:3])
#> mvp object algebraically equal to
#> -27 a^2 c^2 x^2  +  400 a^4 b c^7
deriv(S, rev(letters[1:3]))  # should be the same.
#> mvp object algebraically equal to
#> -27 a^2 c^2 x^2  +  400 a^4 b c^7
```

Also a slightly different form: `aderiv()`, here used to evaluate $\frac{\partial^6 S}{\partial a^3 \partial b \partial c^2}$:

```
aderiv(S, a = 3, b = 1, c = 2)
#> mvp object algebraically equal to
#> 33600 a^2 b c^6  -  108 c x^2
```

Again, pipes work quite nicely:

```
S %<>% aderiv(a=1,b=2) %>% subs(c="x^4") %>% `+`(as.mvp("o^99"))
S
#> mvp object algebraically equal to
#> 50 a^4 x^32  +  o^99
```

Taylor series

The package includes functionality to deal with Taylor and Laurent series:

```
(X <- as.mvp("1+x+x^2 y")^3)
#> mvp object algebraically equal to
#> 1  +  3 x  +  3 x^2  +  3 x^2 y  +  x^3  +  6 x^3 y  +  3 x^4 y  +  3 x^4 y^2  +  3 x^5 y^2  +  x^6
taylor(X,3)         # terms with total power <= 3
#> mvp object algebraically equal to
#> 1  +  3 x  +  3 x^2  +  3 x^2 y  +  x^3
taylor(X,3,"x")     # terms with power of x <= 3
#> mvp object algebraically equal to
#> 1  +  3 x  +  3 x^2  +  3 x^2 y  +  x^3  +  6 x^3 y
onevarpow(X,3,"x") # terms with power of x == 3
#> mvp object algebraically equal to
#> 1  +  6 y
```

```
## second order taylor expansion of f(x)=sin(x+y) for x=1.1, about x=1:
sinxpy <- horner("x+y",c(0,1,0,-1/6,0,+1/120,0,-1/5040))  # sin(x+y)
dx <- as.mvp("dx")
t2 <- sinxpy  + aderiv(sinxpy,x=1)*dx + aderiv(sinxpy,x=2)*dx^2/2
(t2 %<>% subs(x=1,dx=0.1))  # (Taylor expansion of sin(y+1.1), left in symbolic form)
#> mvp object algebraically equal to
#> 0.8912877  +  0.4534028 y  -  0.4458333 y^2  -  0.07597222 y^3  +  0.03659722 y^4  +  0.003291667 y^
(t2 %>% subs(y=0.3))  - sin(1.4)  # numeric; should be small
#> [1] -1.416914e-05
```

Negative powers

The `mvp` package handles negative powers, although the idiom is not perfect and I'm still working on it. There is the `invert()` function:

```
(p <- as.mvp("1+x+x^2 y"))
#> mvp object algebraically equal to
#> 1  +  x  +  x^2 y
invert(p)
#> mvp object algebraically equal to
#> 1  +  x^-2 y^-1  +  x^-1
```

In the above, `p` is a regular multivariate polynomial which includes negative powers. It obeys the same arithmetic rules as other mvp objects:

```
p + as.mvp("z^6")
#> mvp object algebraically equal to
#> 1  +  x  +  x^2 y  +  z^6
```

We can see the generating function for a chess knight:

```
knight(2)
#> mvp object algebraically equal to
#> a^-2 b^-1  +  a^-2 b  +  a^-1 b^-2  +  a^-1 b^2  +  a b^-2  +  a b^2  +  a^2 b^-1  +  a^2 b
```

How many ways are there for a 4D knight to return to its starting square after four moves? Answer:

```
constant(knight(4)^4)
#> [1] 12528
```

Some timings

I will show some timings using a particularly favourable example that exploits the symbolic nature of the `mvp` package.

```
library("spray")
library("mpoly")
n <- 100
k <- kahle(n, r = 3, p = 1:3, symbols = paste0("x", sprintf("%03d", 1:n)))
```

In the above, polynomial `k` has 500 terms of the form xy^2z^3. Coercing `k` to `spray` form would need a 500×500 matrix for the indices, almost every element of which would be zero. This makes the **spray** package slower:

```
library("microbenchmark")

spray_k <- mvp_to_spray(k)
microbenchmark( k^2, spray_k^2 )
#> Unit: milliseconds
#>       expr       min        lq      mean    median        uq       max neval
#>        k^2 102.37778 106.93744 118.4194 113.17985 123.32960 207.2098   100
#>  spray_k^2  46.25651  51.53271  59.3037  55.50252  62.66123 135.0715   100
```

In the above, the first line uses `mvp` functionality, and the second line uses `spray` functionality. The speedup increases for larger polynomials.

References

Kahle, D. 2013. "Mpoly: Multivariate Polynomials in R." *R Journal* 5 (1):162.