arXiv:2603.15156v1 [cs.SC] 16 Mar 2026

Introducing the vfunc R package

A Preprint

RobinK.S. Hankin*斯特灵大学
hankin.robin@gmail.com

2026年3月17日

摘要

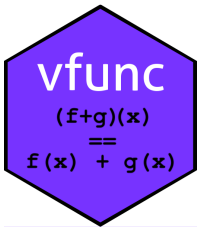
在数学中，给定两个函数 f, g: $\mathbb{R} \longrightarrow \mathbb{R}$，自然可以将 $f+g$ 定义为将 $x \in \mathbb{R}$ 映射到 $f(x) + g(x)$ 的函数。然而，在基础 R 中，函数类的对象并未定义算术方法，因此诸如“f + g”这样的惯用写法会返回错误，尽管它完全符合合理预期。vfunc 包提供了这一功能。此外，它还提供了其他类似特性，从而形成简洁易读的惯用写法，并消除大量编码错误。

1 引言

请考虑以下 R 会话：

```
> f <- function(x){x^2}
> g <- function(x){1/(1-x)}
> f + g
```

f +g 中出错：二元运算符的参数不是数值型



在上文中，对于“f + g”有一个相当明确的预期：它应返回一个函数，该函数返回 f() 与 g() 的和；类似于 function(x){f(x) + g(x)}。然而，它会返回错误，因为 f 和 g 是 S4 类 function 的对象，而该类没有加法方法。此外，在这种情况下无法定义 Arith 组的 S4 方法 [，重载加法“+”]，使这一惯用写法按预期运行。这是因为 function 类在 S4 中是*sealed* 的：禁止为其定义新方法。这里介绍 vfunc R [2] 包，它提供了适用的惯用写法。该包可在 CRAN 上通过 https://CRAN.R-project.org/package=vfunc 获取。该包定义了一个新的 S4 类 vf（“虚函数”），它继承自 function，但可以为其定义新方法。这一机制为将 Arith 方法应用于函数提供了一些途径。

2 inuse 包

该包旨在使 vf 类的对象作为函数运行，同时支持算术运算，并以透明方式执行这些运算。例如：

```
> library("vfunc")
> f <- as.vf(f)
> g <- as.vf(g)
> (f + g)(1:10)
```

[1] 无穷大 3.00000 8.50000 15.6666724.7500035.80000 48.83333 63.85714[9] 80.87500 99.88889

*工作；娱乐

上文中，我们将 f 和 g 强制转换为 S4 类 vf [（即“虚函数”）]的对象。这类对象定义了算术方法，因此可以进行算术组合；例如，加法会被分派到

function(e1, e2){as.vf(function(...){e1(...) + e2(...)})}

vf 类有一个类型为 "function" 的 .Data 插槽，这意味着该类的对象继承了基类 function 的大部分行为；上文中我们看到，e1 和 e2 可以直接使用它们的参数列表来执行。实际上，这意味着 f+g 的行为符合预期，也表明它还可以通过其他方式使用：

```
> (f + 4*g - f*g)(1:10)
```

```
 [1]    NaN 4.00000 11.50000 20.00000 30.25000 42.40000 56.50000 [8]    72.57143
 90.62500110.66667
```

这种惯用写法的优势主要分为两类。首先，代码可以变得更加简洁；其次，可以防范多种难以发现的错误。现在假设 f() 和 g() 是三元函数，各自接受三个参数，例如：

```
> f <- function(x,y,z){x + x*y - x/z}
> g <- function(x,y,z){x^2 - z}
```

以及 $x = 1.2$、 $y = 1.7$、 $z = 4.3$。基于此，我们希望计算

$$(f(x, y, z) + g(x, y, z))(f(x, y, z) + 4 - 2f(x, y, z)g(x, y, z)).$$

如何在 R 中编写这样的表达式？标准方法是

```
>  x <- 1.2
>  y <- 1.7
>  z <- 4.3
> (f(x,y,z) + g(x,y,z))*(f(x,y,z) + 4 - 2*f(x,y,z)*g(x,y,z))
```

```
[1] 2。411975
```

注意，参数列表 (x,y,z) 被重复指定，这里重复了五次。现在使用 vfunc 包：

```
> f <- as.vf(f)
> g <- as.vf(g)
> ((f + g)*(f + 4 - 2*f*g))(x,y,z)
```

```
[1] 2。411975
```

请注意，该包允许将参数列表“因式分解”，使其只出现一次，从而得到更简洁的代码。正如下例所示，这样也可以说更不容易出错。考虑

$$f(x + z, y + z, f(x, x, y) - g(x, x, y)) + g(x + z, y + z, f(x, x, y) - g(x, x, y))$$

（这类表达式常见于动力系统的研究中）。请注意，函数 f 和 g 需要在嵌套的不同层级，分别使用两组不同的参数进行求值：内层为 $(x,\ x,\ y)$，外层为 $(x + z,\ y + z,\ f(x,\ x,\ y) - g(x,\ x,\ y)$。标准的 R 语言惯用写法是

```
> f(x + z, y + z, f(x, x, y) - g(x, x, y)) + g(x + z, y + z, f(x, x, y) - g(x, x, y))
```

```
[1] 64.04918
```

作者可以证明，在这类表达式中查找错误可能很困难 [很容易在某处把 (x,x,y) 打错，却很难发现这个错误]。不过，vfunc 惯用法会

```
> (f + g)(x + z, y + z, (f - g)(x, x, y))
```

```
[1] 64.04918
```

这种写法无疑更简短，也可以说更加简洁，至少作者认为此类构造的出错概率要低得多。采用这种形式，可以确保在表达式的两个层级中，f() 和 g() 每次调用时所使用的参数都完全相同，因为这些参数只出现了一次。

2.1 重载

再次查看 vf 加法的方法，即

function(e1, e2){as.vf(function(...){e1(...) + e2(...)})}

我们看到， + 运算符用于对 e1() 和 e2() 的返回值求和。这个运算符本身也可以重载，vfunc 包对此能够透明地处理，无论使用 S3 还是 S4。以 onion 包 [1] 为例：

```
> library("onion")
> options("show_onions_compactly" = TRUE)
> f <- as.vf(function(x,y){x + x*y})
> g <- as.vf(function(x,y){x^2 + y})
> (f + g - f*g)(1 + Hj,Hk)
```

```
    Re4+2i+2
j-1k
```

3 原语函数

R 语言包含许多作为 S4 Math 泛型的原语函数，包括 sin() 等三角函数，以及 cumsum() 累积和等其他函数。这些函数根深蒂固，无法轻易修改以支持 vf 类对象。该包定义了原语函数的首字母大写版本，使其能够对其他 vf 类对象进行操作。以 sin() 为例：

```
> vfunc::Sin
```

一个 "vf" 类对象 函数 (x) { sin(x) }
<字节码：0x5b9452ac73c0><环境：
namespace:vfunc>

然后，例如，我们可以将三角函数与用户定义函数结合起来：

```
> fun <- as.vf(function(x){x^2 + 2})
> (fun(Sin) + Sin(fun) - 3*Sin*fun)(0.32)
```

```
[1] 0.9769132
```

上面我们看到，包惯用法被用于计算 $\sin^2(0.32) + 3 + \sin(0.32^2 + 2) - 3 \cdot \sin 0.32 \cdot (0.32^2 + 2)$。在基础 R 中：

```
> fun(sin(0.32)) + sin(fun(0.32)) - 3*sin(0.32)*fun(0.32)
```

```
[1] 0.9769132
```

通过这种构造，可以定义如下复合函数：

```
> j <- as.vf(function(x,y){Cos(x) + Sin(x-y)})
> k <- as.vf(function(x,y){Tan(x) + Log(x+y)})
> l <- as.vf(function(x,y){Sin(x/2) + x^2   })
```

（请注意，函数 j()、k() 和 l() 都是二元的）。然后与

```
> (j + k + l)(Sin + Log, Cos + Exp)(Sin + Tan)(0.4)
```

```
[1] 2.545235
```

这种单阶段惯用法进行比较，其内容如下：

```
[1] 2.545235
```

```
> j(sin(sin(0.4) + tan(0.4)) + log(sin(0.4) + tan(0.4)), cos(sin(0.4) + tan(0.4)) +
+ exp(sin(0.4) + tan(0.4))) + k(sin(sin(0.4) + tan(0.4)) + log(sin(0.4) + tan(0.4)),
+ cos(sin(0.4) + tan(0.4)) + exp(sin(0.4) + tan(0.4)))+ l(sin(sin(0.4) + tan(0.4)) +
+ log(sin(0.4) + tan(0.4)), cos(sin(0.4) + tan(0.4)) + exp(sin(0.4) + tan(0.4)))
```

```
[1] 2.545235
```

以及多阶段惯用法：

```
> A <- function(x,y){j(x,y) + k(x,y) + l(x,y)}
> B <- function(x){sin(x) + log(x)}
> C <- function(x){cos(x) + exp(x)}
> D <- function(x){sin(x) + tan(x)}
> x <- 0.4
> A(B(D(x)), C(D(x)))
```

```
[1] 2.545235
```

请注意，单阶段惯用法非常冗长，而多阶段惯用法不透明 [，并且重复出现 (x,y) 和 x]。

4 结论

vfunc 包允许对函数进行“因式分解”，也就是说，将 f(x) + g(x) 重写为 (f + g)(x)。这使惯用法更简洁，并消除了一类特定的编码错误。该包还支持递归应用这类方法。后续工作可包括评估该包的计算效率。

参考文献

[1] Hankin, R. K. S.（2006年）。使用 R 的赋范除法代数：介绍 onion 包。*R News*，6(2):49–52。[2] R 核心团队（2024年）。*R*：统计计算语言与环境。R 统计计算基金会，奥地利维也纳。