Restrict functions to a smaller domain with `restrict_fun()` in the `doBy` package

Søren Højsgaard

doBy version 4.6.12 as of 2022-02-06

Contents

1 Introduction 1

2 Restrict a functions domain: `restrict_fun()` 1
   2.1 Using an auxillary environment 1
   2.2 Substitute restricted values into function 2

3 Example: Benchmarking 3

1 Introduction

The `doBy` package contains a variety of utility functions. This working document describes some of these functions. The package originally grew out of a need to calculate groupwise summary statistics (much in the spirit of `PROC SUMMARY` of the SAS system), but today the package contains many different utilities.

2 Restrict a functions domain: `restrict_fun()`

The `restrict_fun` function can restrict the domain of a function. For example, if $f(x, y) = x + y$ then $g(x) = f(x, 10)$ is a restriction of $f$ to be a function of $x$ alone.

There are two approaches: 1) Store the restricted arguments in an auxillary environment and 2) substitute the restricted arguments into the function.

2.1 Using an auxillary environment

```r
> f1 <- function(a, b, c=4, d=9){
  a + b + c + d
}
> f1_ <- restrict_fun(f1, list(b=7, d=10))
> class(f1_)
```

## [1] "scaffold"

```r
```
We see the new function is a function of \( a \) and \( c \) with \( c \) being given a default value, but what the function does is not clear. However, it does evaluate correctly:

\[
\texttt{f1} = \texttt{function (a, c = 4)} \quad \texttt{\{ \}
\quad \texttt{args <- arg_getter()} \quad \texttt{\}
\quad \texttt{do.call(fun, args)}
\quad \texttt{\}}
\]

\[
\texttt{\textless{} environment: 0x55767b55dcb0} \texttt{> f1\_}(100)
\]

\[
\texttt{[1]} \texttt{121}
\]

The restricted values are stored in an extra environment in the \texttt{scaffold} object and the original function is stored in the scaffold functions environment:

\[
\texttt{\textgreater{} get_restrictions(f1\_)}
\]

\[
\begin{align*}
\texttt{\$b} & \quad [1] \texttt{7} \\
\texttt{\$d} & \quad [1] \texttt{10}
\end{align*}
\]

\[
\quad \texttt{\textless{} attr(f1\_, \textquoteright{}arg\_env\textquoteright{})\texttt{\&args} \texttt{\# Same result}}
\quad \texttt{get\_fun(f1\_)}
\quad \texttt{\}}
\quad \texttt{\textless{} attr(f1\_, \textquoteright{}arg\_env\textquoteright{})\texttt{\&fun} \texttt{\# Same result}}
\]

Similarly

\[
\texttt{\textgreater{} rnorm5 \textless{} restrict\_fun(rnorm, list(n=5))}
\quad \texttt{rnorm5\_()}
\]

\[
\texttt{[1]} \texttt{1.06144 0.07263 0.46731 -1.24649 -0.41485}
\]

2.2 Substitute restricted values into function

With substitution, it is clear what is happening:

\[
\texttt{\textgreater{} fis\_ \textless{} restrict\_fun\_sub(f1, list(b=7, d=10))}
\quad \texttt{fis\_}
\quad \texttt{\}}
\quad \texttt{a + 7 + c + 10}
\quad \texttt{\}}
\quad \texttt{\}}
\quad \texttt{\}}
\quad \texttt{fis\_}(100)
\]

\[
\texttt{[1]} \texttt{121}
\]

However, absurdities can arise:
> f2 <- function(a) {
>     a <- a + 1
>     a
> }
> ## Notice that the following is absurd
> f2s_ <- restrict_fun_sub(f2, list(a = 10))
> f2s_
> ## function ()
> ## {
> ## 10 <- 10 + 1
> ## 10
> ## }
> # do not run: f2s_()
> try(f2s_())
> ## Error in 10 <- 10 + 1 : invalid (do_set) left-hand side to assignment
> ## Using the environment approach, the result makes sense
> f2_ <- restrict_fun(f2, list(a = 10))
> f2_
> ## function ()
> ## {
> ## args <- arg_getter()
> ## do.call(fun, args)
> ## }
> ## <environment: 0x557674264078>
> f2_()
> ## [1] 11

3 Example: Benchmarking

Consider a simple task: Creating and inverting Toeplitz matrices for increasing dimensions:
> n <- 4
> toeplitz(1:n)

### [1,] 1 2 3 4
### [2,] 2 1 2 3
### [3,] 3 2 1 2
### [4,] 4 3 2 1

A naive implementation is
> inv_toeplitz <- function(n) {
>     solve(toeplitz(1:n))
> }
> inv_toeplitz(4)

### [1,] -0.4 0.5 0.0 0.1
### [2,] 0.5 -1.0 0.5 0.0
### [3,] 0.0 0.5 -1.0 0.5
We can benchmark timing for different values of \( n \) as

```r
> library(microbenchmark)
> microbenchmark(
  inv_toeplitz(4), inv_toeplitz(8), inv_toeplitz(16),
  inv_toeplitz(32), inv_toeplitz(64),
  times=5
)
```

<table>
<thead>
<tr>
<th>expr</th>
<th>min</th>
<th>lq</th>
<th>mean</th>
<th>median</th>
<th>uq</th>
<th>max</th>
<th>neval</th>
<th>cld</th>
</tr>
</thead>
<tbody>
<tr>
<td>inv_toeplitz(4)</td>
<td>39.66</td>
<td>40.65</td>
<td>42.47</td>
<td>41.10</td>
<td>42.90</td>
<td>48.03</td>
<td>5</td>
<td>a</td>
</tr>
<tr>
<td>inv_toeplitz(8)</td>
<td>44.47</td>
<td>47.59</td>
<td>49.46</td>
<td>48.07</td>
<td>48.79</td>
<td>58.40</td>
<td>5</td>
<td>a</td>
</tr>
<tr>
<td>inv_toeplitz(16)</td>
<td>63.81</td>
<td>63.85</td>
<td>68.32</td>
<td>65.69</td>
<td>66.46</td>
<td>81.81</td>
<td>5</td>
<td>a</td>
</tr>
<tr>
<td>inv_toeplitz(32)</td>
<td>126.00</td>
<td>126.65</td>
<td>665.65</td>
<td>130.76</td>
<td>131.88</td>
<td>2812.96</td>
<td>5</td>
<td>a</td>
</tr>
<tr>
<td>inv_toeplitz(64)</td>
<td>399.94</td>
<td>407.06</td>
<td>420.14</td>
<td>416.49</td>
<td>420.60</td>
<td>456.62</td>
<td>5</td>
<td>a</td>
</tr>
</tbody>
</table>

However, it is tedious (and hence error prone) to write these function calls.

A programmatic approach using `restrict_fun` is as follows: First create list of scaffold objects:

```r
> n.vec <- c(4, 8, 16, 32, 64)
> scaf.list <- lapply(n.vec,
                     function(ni){
                       restrict_fun(inv_toeplitz, list(n=ni))
                     })
```

Each element is a function (a scaffold object, to be precise) and we can evaluate each / all functions as:

```r
> scaf.list[[1]]
## function ()
## {
##   args <- arg_getter()
##   do.call(fun, args)
## }
## <environment: 0x557678c53b70>
```

To use the list of functions in connection with microbenchmark we bquote all functions using

```r
> bquote_list <- function(fnlist){
    lapply(fnlist, function(g) {
      bquote(.g())
    })
}
```

We get:
To use microbenchmark we must name the elements of the list:

```r
names(bq.list) <- n.vec
microbenchmark(
  list = bq.list,
  times = 5
)
```

To summarize: to experiment with many difference values of \(n\) we can do

```r
n.vec <- seq(50, 700, by=50)
scaf.list <- lapply(n.vec,
  function(ni){
    restrict_fun(inv_toeplitz, list(n=ni))
  })
bq.list <- bquote_list(scaf.list)
names(bq.list) <- n.vec
mb <- microbenchmark(
  list = bq.list,
  times = 5
)
doBy::mb_summary(mb) %>% head(4)
```

Notice: Above, doBy::mb_summary is a faster version of the summary method for microbenchmark objects than the method provided by the microbenchmark package.
```r
> par(mfrow=c(1,2))
> y <- mb_summary(mb)$mean
> plot(n.vec, y)
> plot(log(n.vec), log(y))
> mm <- lm(log(y) ~ log(n.vec))
> broom::tidy(mm)

## # A tibble: 2 x 5
## #   term     estimate std.error statistic p.value
## #(chr)      <dbl>      <dbl>      <dbl>    <dbl>
## 1 (Intercept) -4.39     0.359      -12.2 3.97e-8
## 2 log(n.vec)  2.63      0.0624     42.1 2.08e-14

> abline(mm)
```