Section functions to a smaller domain with `section_fun()` in the `doBy` package

Søren Højsgaard

doBy version 4.6.20 as of 2023-11-01

Contents

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 Section a functions domain: `section_fun()`

Let E be a subset of the cartesian product $X \times Y$ where X and Y are some sets. Consider a function $f(x, y)$ defined on E. Then for any $x \in X$, the section of E defined by x (denoted E_x) is the set of y's in Y such that (x, y) is in E, i.e.

$$E_x = \{y \in Y | (x, y) \in E\}$$

Correspondingly, the section of $f(x, y)$ defined by x is the function f_x defined on E_x given by $f_x(y) = f(x, y)$.

For example, if $f(x, y) = x + y$ then $f_x(y) = f(10, y)$ is a section of f to be a function of y alone.

There are two approaches: 1) replace the section values in the function (default) or 2) store the section values in an auxiliary environment.

2.1 Replace section into function body

Default is to replace section value in functions body:

```
> f  <- function(a, b, c=4, d=9){
      a + b + c + d
  }
> fr_ <- section_fun(f, list(b=7, d=10))
> fr_

## function (a, c = 4, b = 7, d = 10)
## {
##     a + b + c + d
## }

> f(a=10, b=7, c=5, d=10)

## [1] 32

> fr_(a=10, c=5)

## [1] 32
```

2.2 Using an auxiliary environment

An alternative is to store the section values in an auxiliary environment:

```
> fe_ <- section_fun(f, list(b=7, d=10), method = "env")
> fe_

## function (a, c = 4)
## {
##     . <- "use get_section(function_name) to see section"
##     . <- "use get_fun(function_name) to see original function"
##     args <- arg_getter()
##     do.call(fun, args)
## }
## <environment: 0x56036ae3cda8>

> f(a=10, b=7, c=5, d=10)

## [1] 32

> fe_(a=10, c=5)

## [1] 32
```

The section values are stored in an extra environment in the `scaffold` object and the original function is stored in the scaffold functions environment:

```
> get_section(fe_)

## $b
## [1] 7
##
## $d
## [1] 10

> ## attr(fe_, "arg_env")$args ## Same result
> get_fun(fe_)

## function(a, b, c=4, d=9){
##     a + b + c + d
## }
## <bytecode: 0x5603695917a8>

> ## environment(fe_)$fun ## Same result
```

3 Example: Benchmarking

Consider a simple task: Creating and inverting Toeplitz matrices for increasing dimensions:

```
> n <- 4
> toeplitz(1:n)

##      [,1] [,2] [,3] [,4]
## [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]       [,2] [,3] [,4]
## [1,] -0.4  5.000e-01  0.0  0.1
```

```
## [2,]  0.5 -1.000e+00  0.5  0.0
## [3,]  0.0  5.000e-01 -1.0  0.5
## [4,]  0.1 -6.939e-18  0.5 -0.4
```

We can benchmark timing for different values of n as

```
> library(microbenchmark)

## Warning:  package 'microbenchmark' was built under R version 4.4.0

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

## Unit: microseconds
##              expr    min     lq   mean median     uq     max neval cld
##   inv_toeplitz(4)  15.66  16.42  16.88  16.69  16.90   18.71     5   a
##   inv_toeplitz(8)  18.84  18.92  21.88  19.85  21.32   30.44     5   a
##  inv_toeplitz(16)  23.82  24.17  29.89  25.66  26.03   49.77     5   a
##  inv_toeplitz(32)  45.62  46.52 348.97  49.60 102.54 1500.60     5   a
##  inv_toeplitz(64) 118.00 123.88 481.43 126.63 132.14 1906.52     5   a
```

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

A programmatic approach using `section_fun` is as follows: First create a list of sectioned functions:

```
> n.vec  <- c(4, 8, 16, 32, 64)
> fun_list <- lapply(n.vec,
                     function(ni){
                         section_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:

```
> fun_list[[1]]

## function (n = 4)
## {
##     solve(toeplitz(1:n))
## }

> fun_list[[1]]()
```

```
##      [,1]       [,2] [,3] [,4]
## [1,] -0.4  5.000e-01  0.0  0.1
## [2,]  0.5 -1.000e+00  0.5  0.0
## [3,]  0.0  5.000e-01 -1.0  0.5
## [4,]  0.1 -6.939e-18  0.5 -0.4
```

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

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

We get:

```
> bq_fun_list <- bquote_list(fun_list)
> bq_fun_list[[1]]

## (function (n = 4)
## {
##     solve(toeplitz(1:n))
## })()

> ## Evaluate one:
> eval(bq_fun_list[[1]])

##      [,1]       [,2] [,3] [,4]
## [1,] -0.4  5.000e-01  0.0  0.1
## [2,]  0.5 -1.000e+00  0.5  0.0
## [3,]  0.0  5.000e-01 -1.0  0.5
## [4,]  0.1 -6.939e-18  0.5 -0.4

> ## Evaluate all:
> ## sapply(bq_fun_list, eval)
```

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

```
> names(bq_fun_list) <- n.vec
> microbenchmark(
    list  = bq_fun_list,
    times = 5
  )
```

```
## Unit: microseconds
##  expr    min     lq   mean median     uq    max neval cld
##     4  17.06  17.12  17.80  17.34  18.68  18.83     5  a 
##     8  19.30  20.16  21.59  21.12  22.43  24.96     5  a 
##    16  24.21  25.00  87.71  25.08  31.05 333.20     5  ab
##    32  46.30  46.55  55.48  47.63  49.65  87.29     5  ab
##    64 120.22 132.54 170.65 133.76 149.59 317.16     5   b
```

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