Colour Schemes

Barry Rowlingson

May 28, 2009

R has several functions that produce or manipulate colours. It also has functions that produce or manipulate vectors of colours (colour palettes), and some functions for producing colour ramps. But these functions do not take care of the final step of producing a graphic from your data - that of controlling which colour comes from what data value.

This package aims to fill that gap by providing the means to create *colour scheme* functions. A colour scheme function is an R function that takes data values as argument and produces colours as an output. Access the package functions in the usual way with the R `library` function:

```
> library(colourschemes)
```

The package provides constructors that return colour scheme functions. The general way of working is like this:

```
> sc = specificScheme(data,colours,etc)
> c1 = sc(data)
> c2 = sc(newdata)
```

Firstly you construct a function (`sc` here) using a specific colour scheme function. The constructor may take parameters that are data values, colours, and other things. Secondly you apply the constructed function to some data - either the existing data or new data. At this point the function is fixed according to the scheme and the parameters, and the returned colours are only dependent on the argument. This way you can be sure that the mapping of data values to colours is constant.

A simple colour scheme function is one that produces a colour for each level of a factor. The function `factorScheme` does that:

```
> f = factor(c("a", "a", "b", "a", "c"))
> fs = factorScheme(f, c("red", "blue", "green"))
> fs(f)
```

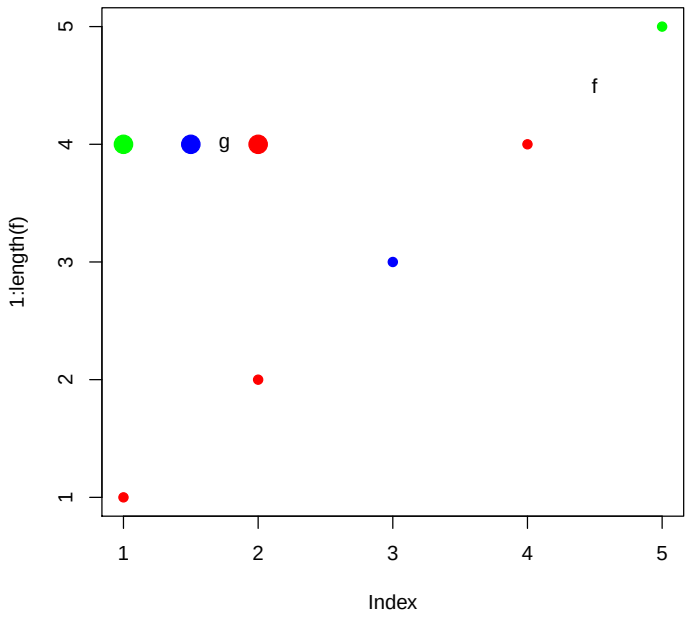
```
    a     a     b     a     c
"red" "red" "blue" "red" "green"
```

The function `fs` can be re-used with any data that has the same levels as the original source, to guarantee that the same colours are used:

```
> g = factor(c("c", "b", "a"))
> fs(g)

      c       b       a
"green"  "blue"   "red"

> plot(1:length(f), col = fs(f), pch = 19)
> ng = length(g)
> points(seq(1, 2, len = ng), rep(4, ng), col = fs(g), pch = 20,
+     cex = 3)
> text(4.5, 4.5, "f")
> text(1.75, 4, "g")
```



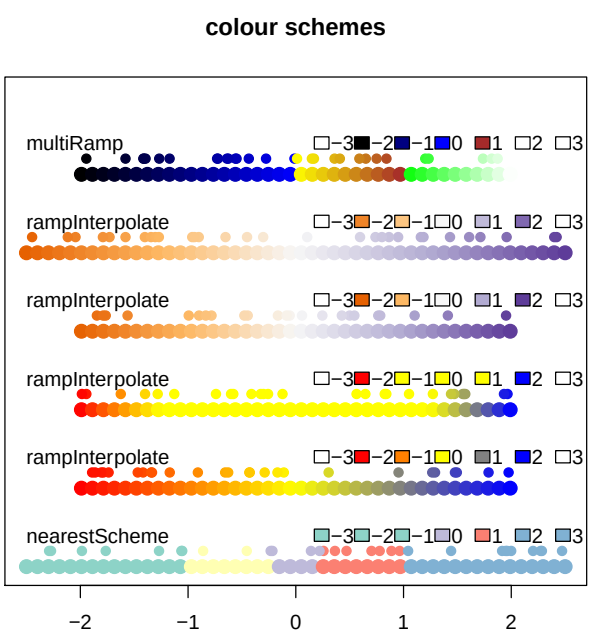
That's about as much as you can do with categorical data - a one-to-one mapping of levels to colours. Continuous data has more possibilities.

The *nearest* scheme takes a data frame of values and colours and produces a colour scheme function that maps data values to the colour corresponding to the nearest value in the data frame.

```
> ns = nearestScheme(data.frame(values = c(0, 1, 2), col = c("red",
+     "blue", "green")))
> s = seq(-0.5, 2.5, by = 0.5)
> rbind(s, ns(s))
```

```
  [,1]   [,2]  [,3]  [,4]   [,5]   [,6]    [,7]
s "-0.5" "0"   "0.5" "1"    "1.5"  "2"     "2.5"
  "red"  "red" "red" "blue" "blue" "green" "green"
```

```
> set.seed(123)
> require(RColorBrewer)
> x = seq(-2.5, 2.5, len = 50)
> srange = max(abs(range(x))) * c(-1, 1)
> schemes = list()
> schemes[[length(schemes) + 1]] = nearestScheme(data.frame(values = c(-1.5,
+     -0.5, 0, 0.5, 1.5), col = brewer.pal(5, "Set3")))
> schemes[[length(schemes) + 1]] = rampInterpolate(c(-2, 2), c("red",
+     "yellow", "blue"))
> schemes[[length(schemes) + 1]] = rampInterpolate(c(-2, 2), c("red",
+     "yellow", "blue"), interpolate = "spline", bias = 0.3)
> schemes[[length(schemes) + 1]] = rampInterpolate(c(-2, 2), brewer.pal(5,
+     "PuOr"))
> schemes[[length(schemes) + 1]] = rampInterpolate(srange, brewer.pal(5,
+     "PuOr"))
> schemes[[length(schemes) + 1]] = multiRamp(rbind(c(-2, 0), c(0,
+     1), c(1, 2)), list(c("black", "blue"), c("yellow", "brown"),
+     c("green", "white")))
> nt = length(schemes)
> plot(srange, c(1, nt + 1), type = "n", xlab = "", ylab = "",
+     axes = FALSE, main = "colour schemes")
> axis(1)
> box()
> for (i in 1:nt) {
+     points(x, rep(i, length(x)), col = schemes[[i]](x), pch = 19,
+         cex = 1.5)
+     for (k in 1:1) {
+         xr = runif(30, min(x), max(x))
+         points(xr, rep(i, length(xr)) + 0.2 * k, col = schemes[[i]](xr),
+             pch = 19, cex = 1)
+     }
+     text(min(x), i + 0.4, class(schemes[[i]])[1], adj = c(0,
+         0.5))
+     legend(par()$usr[2], i + 0.4, xjust = 1, yjust = 0.5, as.character(pretty(x)),
+         fill = schemes[[i]](pretty(x)), horiz = TRUE, bty = "n",
+         cex = 1, x.intersp = 0.1, text.width = 0.05)
+ }
```


colour schemes
multiRamp
rampInterpolate
rampInterpolate
rampInterpolate
rampInterpolate
nearestScheme
−3 −2 −1 0 1 2 3
−2 −1 0 1 2