Object Oriented Microarray and Proteomics Analysis (OOMPA)

Kevin R. Coombes

December 28, 2010

Contents

1 Introduction

OOMPA is a suite of object-oriented tools for processing and analyzing large biological data sets, such as those arising from mRNA expression microarrays or mass spectrometry proteomics.

This vignette documents the base package, *oompaBase*. A rcitical (but invisible to the user) feature of the *oompaBase* package is that it defines a `class union` allowing you to use "numeric" or "NULL" objects in the design of an S4 class. More intersting user-visibel features include alternative color schemes and vectorized matrix operations to speed the computation of row-by-row mweans, variances, and t-tests.

2 Getting Started

You invoke the package in the usual way:

```
> library(oompaBase)
```

3 Color Schemes

To illustrate the various color schemes, we first create a structured matrix:

```
> mat <- matrix(1:1024, ncol = 1)
```

The following code is used to generate Figure 1.

```
> opar <- par(mfrow = c(5, 1), mai = c(0.3, 0.5, 0.2, 0.2))
> image(mat, col = redgreen(64), main = "redgreen")
> image(mat, col = jetColors(128), main = "jetColors")
> image(mat, col = blueyellow(32), main = "blueyellow")
> image(mat, col = redscale(64), main = "redscale")
> image(mat, col = bluescale(64), main = "bluescale")
> par(opar)
```

4 Row-by-row Matrix Operations

We now want to illustrate the "matrix operations" that allow for rapid computation of row-by-row means, variances, and t-tests.

We start by creating a slightly more interesting matrix full of random data. First, we make the variance larger in the second half (by column) of the data than in the first half.

```
> ng <- 10000
> ns <- 50
> dat <- matrix(rnorm(ng * ns, 0, rep(c(1, 2), each = 25)), ncol = ns,
+     byrow = TRUE)
```

Next, we shift the mean for te first 500 "genes" (rows).

```
> dat[1:500, 1:25] <- dat[1:500, 1:25] + 2
```

In order to compute t-tests, we also assign arbitrary labels separating the "sample columns" into two groups.

```
> clas <- factor(rep(c("Good", "Bad"), each = 25))
```

Here we compute the row-by-row means.

```
> a0 <- proc.time()
> myMean <- matrixMean(dat)
> used0 <- proc.time() - a0
```

For comparions purposes, we perfom the same computation using `apply`.

```
> a1 <- proc.time()
> mm <- apply(dat, 1, mean)
> used1 <- proc.time() - a1
```

The results are the same, to within round-off error.

```
> summary(as.vector(myMean - mm))
```

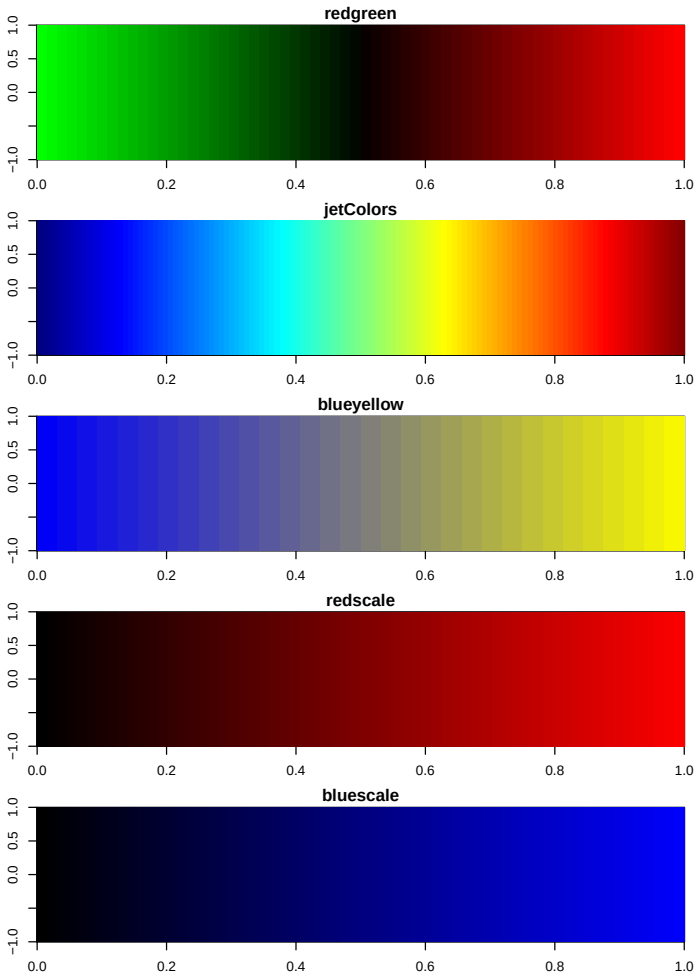

Figure 1: Five color schemes.

```
      Min.    1st Qu.     Median       Mean    3rd Qu.       Max.
-6.661e-16 -2.776e-17  0.000e+00 -8.805e-19  2.776e-17  8.882e-16
```

There is a measurable (although not really user-perceptible) difference in the time for the two methods.

```
> used0

   user  system elapsed
   0.00    0.00    0.02

> used1

   user  system elapsed
   0.17    0.00    0.17
```

Here we compute the variances using two different methods.

```
> a0 <- proc.time()
> myVar <- matrixVar(dat, myMean)
> a1 <- proc.time()
> vv <- apply(dat, 1, var)
> a2 <- proc.time()
```

Again, the values are the same:

```
> summary(as.vector(myVar - vv))

      Min.    1st Qu.     Median       Mean    3rd Qu.       Max.
-4.441e-15 -4.441e-16  0.000e+00  6.362e-18  4.441e-16  3.553e-15
```

However, the time savings is substantially larger.

```
> a1 - a0

   user  system elapsed
      0       0       0

> a2 - a1

   user  system elapsed
   0.41    0.00    0.40
```

Not surprisingly, there is an even bigger time savings when computing (equal variance) t-statistics.

```
> t0 <- proc.time()
> myT <- matrixT(dat, clas)
> t1 <- proc.time()
> tt <- sapply(1:nrow(dat), function(i) {
+     t.test(dat[i, clas == "Bad"], dat[i, clas == "Good"], var.equal = T)$statistic
+ })
> t2 <- proc.time()
```

```
> summary(as.vector(tt - myT))

      Min.    1st Qu.     Median       Mean    3rd Qu.       Max.
-4.441e-15 -1.110e-16  0.000e+00  3.873e-18  1.110e-16  5.329e-15

> t1 - t0

   user  system elapsed
   0.01    0.00    0.03

> t2 - t1

   user  system elapsed
   4.42    0.00    4.42
```