Using the bias-corrected selection frequency to map eQTL

Jake Michaelson

November 24, 2010

1 Simulating data

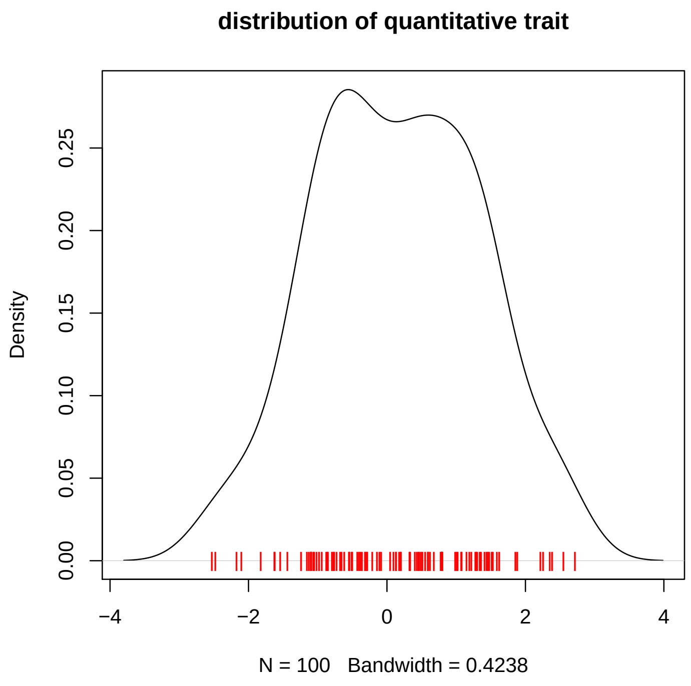
All that is needed to map eQTL with Random Forests is a matrix of genotypes and a vector of gene expression values. Although there are many publicly available expression and genotype data sets (see the GeneNetwork website, for instance), for convenience, we simulate these data here.

First we simulate the genotypes for 100 individuals:

```
> source("functions.R")
> set.seed(2341)
> x = simgeno(100)
```

Then we simulate a trait that is the additive effect of three loci (the 200th marker, the 500th marker, and the 750th marker):

```
> set.seed(213756)
> y = simtrait(x[, 200], 1, 0.4) + simtrait(x[, 500], 0.75) - simtrait(x[,
+     750], 1.5, 0.3)
> pdens(y, main = "distribution of quantitative trait")
```



2 Mapping eQTL with RFSF

Next we map the eQTL with Random Forests, and extract the selection frequencies:

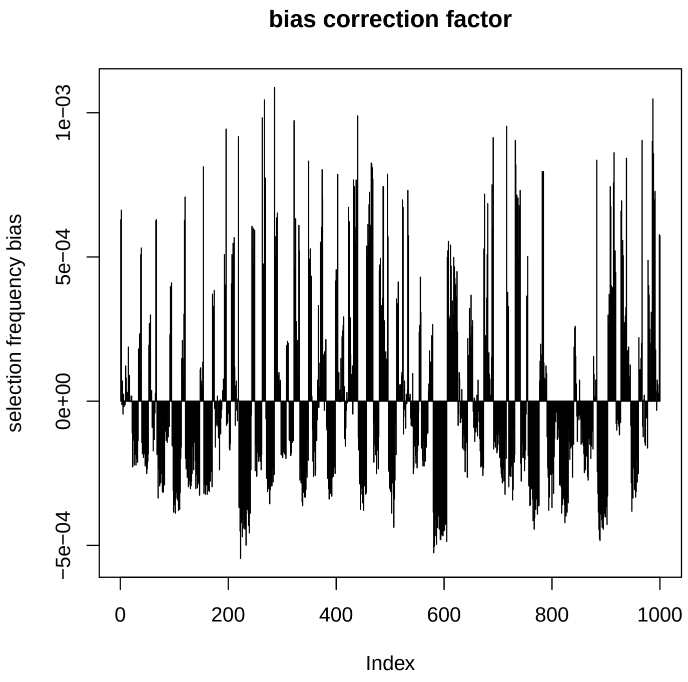
```
> library(randomForest)
> rf = randomForest(y = y, x = x, ntree = 2000)
> sf = rfsf(rf)
```

3 Estimating and accounting for selection bias

Now we estimate the marker-specific selection bias when we know the null hypothesis to be true (no association between genotypes and trait). The more trees we use, the more stable the estimate — here we use 10,000 trees. Also note that whatever additional or non-default arguments (e.g. `mtry` or `nodesize`) are used when mapping should also be passed to `estBias()`.

```
> corr = estBias(x, 10000, verbose = F)
```

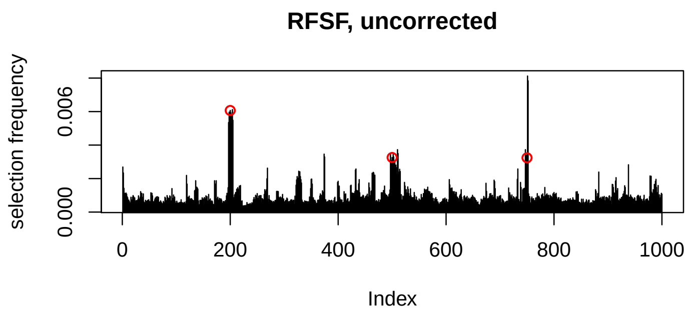
```
> plot(corr, type = "h", ylab = "selection frequency bias",
+     main = "bias correction factor")
```

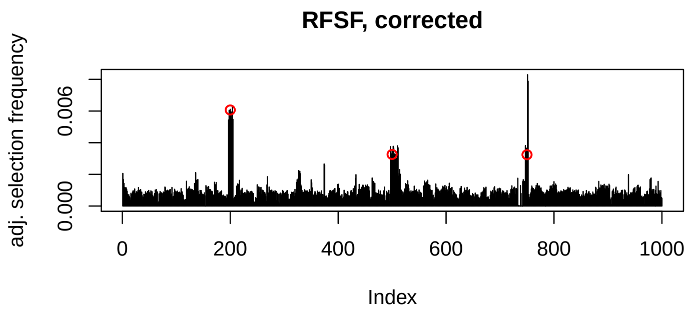


At this point we can just subtract `corr` from `sf` to get the corrected selection frequencies. Notice the peaks that vanish after the correction. Here we have fit a synthetic trait with a moderate amount of noise. When fitting traits with a high signal to noise ratio, the effect of bias (and hence its correction) is less noticable — the background effects of the bias are very small in comparison to the signal.

```
> sf.corr = sf - corr
> sf.corr[sf.corr < 0] = 0

> par(mfrow = c(2, 1))
> plot(sf, type = "h", ylab = "selection frequency", main = "RFSF, uncorrected")
> points(c(200, 500, 750), sf[c(200, 500, 750)], col = "red", lwd = 1.5)
> plot(sf.corr, type = "h", ylab = "adj. selection frequency",
+     main = "RFSF, corrected")
> points(c(200, 500, 750), sf[c(200, 500, 750)], col = "red", lwd = 1.5)
```





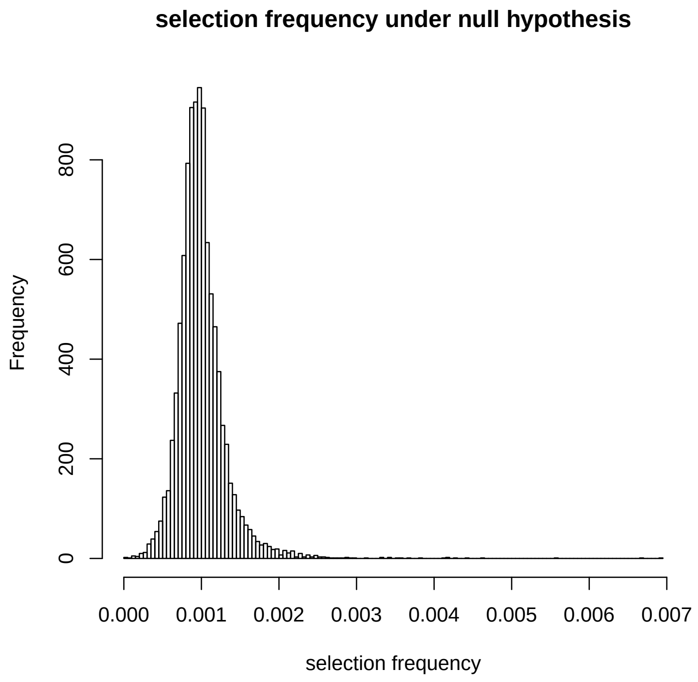
The estimated selection bias is independent of the response — it is only a function of the predictors (genotypes). This means that the same bias correction can be used for each new expression trait — it does not need to be re-estimated.

4 Significance

The bias-adjusted RFSF can be useful as-is for ranking markers for further analysis. However, if *P* values are desired, a few more steps must be taken. First, we need to get an idea of what kind of selection frequencies we can expect under the null hypothesis:

```
> sf.null = numeric()
> for (i in 1:10) {
+     rf.null = randomForest(y = sample(y), x = x, ntree = 2000)
+     tmp = rfsf(rf.null) - corr
+     tmp[tmp < 0] = 0
+     sf.null = c(sf.null, tmp)
+     print(i)
+ }

> hist(sf.null, main = "selection frequency under null hypothesis",
+     xlab = "selection frequency", breaks = 100)
```



We now have an empirical null distribution of the selection frequencies. Using the `ecdf()` function, we can get estimates of *P* values. Note, however, that since we only have 10,000 data points in our null distribution, any *P* value less than 0.0001 will be 0. In other words, we cannot estimate *P* values between 0.0001 and 0 without adding more data points to the null distribution. An alternative might be to fit a parametric distribution (such as a mixture of beta densities) to the empirical null distribution, but for simplicity's sake, we'll use the empirical null distribution to get *P* values.

```
> pnull = ecdf(sf.null)
> P = 1 - pnull(sf)
```

Adjust for multiple testing and check what the FDR is at our "causal markers" (your numbers may vary since we did not use `set.seed()` when generating the null distribution).

```
> Q = p.adjust(P, "fdr")
> sum(Q < 0.05)

[1] 11

> Q[c(200, 500, 750)]

[1] 0.02727273 0.06923077 0.06923077
```

5 Running RF in parallel

Running RF over tens of thousands of expression traits can be time consuming. However, it is quite straightforward to take advantage of multi-core hardware to make the problem tractable. We use the `snow` package to create a cluster that distributes the workload over several processors or cores.

```
> library(snow)
> cl = makeSOCKcluster(8)
> clusterExport(cl, "rfsf")
```

Next, we write a small wrapper function that will run RF given a response (expression) and a matrix of predictors (markers), and then return the selection frequencies.

```
> ff = function(y, x, ntree) {
+     rf = randomForest::randomForest(y = y, x = x, ntree = ntree)
+     sf = rfsf(rf)
+ }
```

As an example here, we create a toy matrix `expr` of expression traits, where each row corresponds to a gene and the columns to individuals.

```
> expr = matrix(rnorm(8 * 100), 8, 100)
```

Finally, we use the `parApply` function to parallelize the mapping of eQTL with RF. This distributes the mapping tasks as evenly as possible across the available processors (which were determined when we called `makeSOCKcluster`).

```
> sf = parApply(cl, expr, 1, ff, x, 1000)
```

Now that we have the selection frequencies, we apply the bias correction:

```
> sf = t(sf - corr)
```