Imputed SNP analyses and meta-analysis with snpMatrix

David Clayton

October 23, 2008

Getting started

The need for imputation in SNP analysis studies occurs when we have a smaller set of samples in which a large number of SNPs have been typed, and a larger set of samples typed in only a subset of the SNPs. We use the smaller, complete dataset (which will be termed the *training dataset*) to impute the missing SNPs in the larger, incomplete dataset (the *target dataset*). Examples of such applications include:

- use of HapMap data to impute association tests for a large number of SNPs, given data from genome-wide studies using, for example, a 500K SNP array, and
- meta-analyses which seek to combine results from two platforms such as the Affymetrix 500K and Illumina 550K platforms.

Here we will not use a real example such as the above to explore the use of `snpMatrix` for imputation, but generate a fictitious example using the data analysed in earlier exercises. This is particularly artificial in that we have seen that these data suffer from extreme heterogeneity of population structure.

We start by attaching the required libraries and accessing the data used in the exercises:

```
> library(snpMatrix)
> library(hexbin)
> data(for.exercise)
```

Next we select alternate SNPs to be potentially missing or present in the target dataset:

```
> sel <- seq(1, ncol(snps.10), 2)
> missing <- snps.10[, sel]
> present <- snps.10[, -sel]
> missing

A snp.matrix with  1000 rows and  14251 columns
Row names:  jpt.869 ... ceu.464
Col names:  rs7909677 ... rs12218790
```

```
> present
```

```
A snp.matrix with  1000 rows and  14250 columns
Row names:  jpt.869 ... ceu.464
Col names:  rs7093061 ... rs7899159
```

We also need to know where the SNPs are on the chromosome in order to avoid having to search the entire chromosome for suitable predictors of a missing SNP:

```
> pos.miss <- snp.support$position[sel]
> pos.pres <- snp.support$position[-sel]
```

Calculating the imputation rules

The next step is to calculate a set of regression equations which provide rules for imputing the `missing` SNPs from the `present` SNPs. This is carried out by the function `snp.imputation`:

```
> rules <- snp.imputation(present, missing, pos.pres, pos.miss)
```

This took a short while. But the wait was really quite short when we consider what the function has done. For each of the 14,251 SNPs in the "missing" set, the function has performed a forward step-wise regression on the 50 nearest SNPs in the "present" set, stopping each search either when the R^2 for prediction exceeds 0.9 or after including 4 SNPs in the regression. The figures 50, 0.9 and 4 in the previous sentence are the default values of the function arguments `try`, `r2.stop`, and `max.X`. Each element of `rules` contains a regression equation together with the R^2 achieved:

```
> rules[1]
```

```
rs7909677 ~ rs2496276+rs4881551+rs4880750+rs9419498 (MAF = 0.0550505, R-squared = 0
```

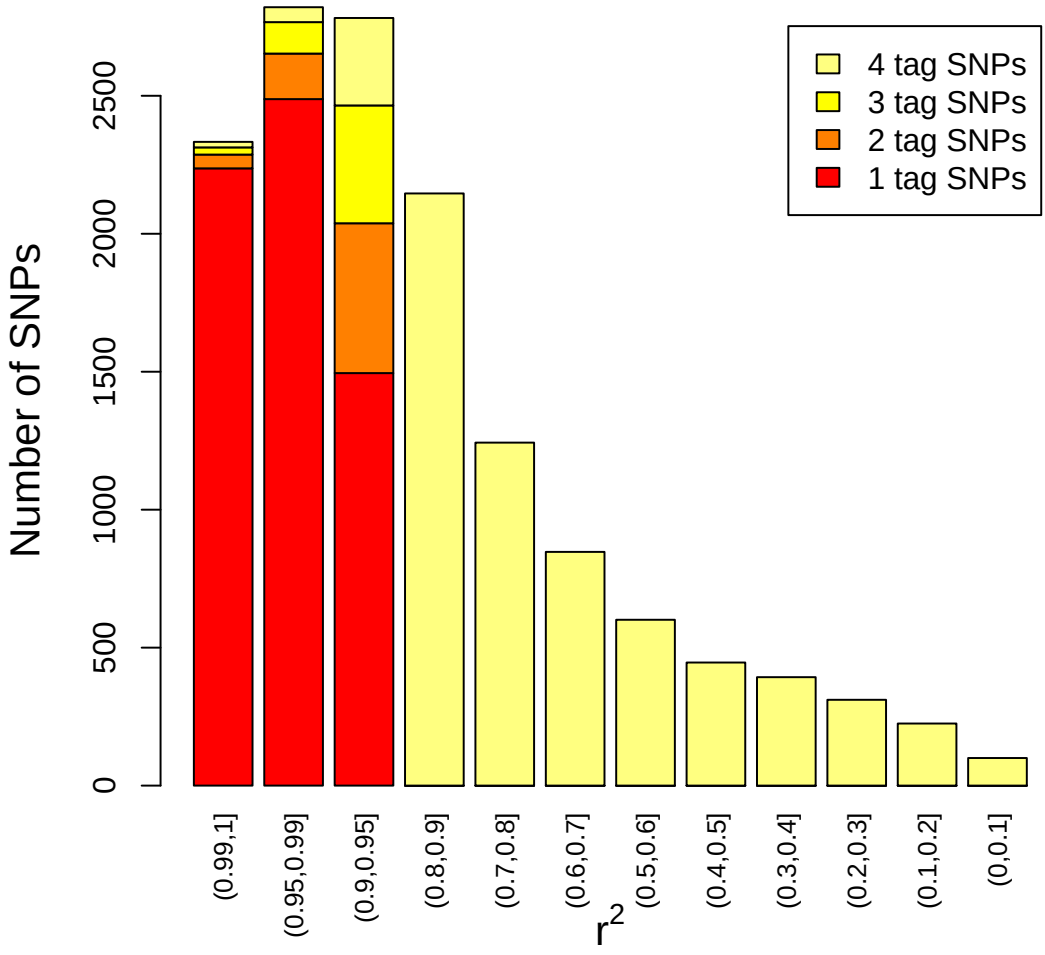
A summary table of all the 14,251 regression equations is generated by

```
> summary(rules)
```

```
             SNPs used
R-squared        1    2    3    4
  (0,0.1]        0    0    0  100
  (0.1,0.2]      0    0    0  225
  (0.2,0.3]      0    0    0  311
  (0.3,0.4]      0    0    0  393
  (0.4,0.5]      0    0    0  446
  (0.5,0.6]      0    0    0  601
  (0.6,0.7]      0    0    0  847
  (0.7,0.8]      0    0    0 1243
  (0.8,0.9]      0    0    0 2146
  (0.9,0.95]  1495  543  427  317
  (0.95,0.99] 2488  165  114   54
  (0.99,1]    2237   50   26   20
```

Columns represent the number of SNPs in the regression and rows represent grouping on R^2. The first column (headed `0`) represents SNPs which were monomorphic in the sample. The same information may be displayed graphically by

```
> plot(rules)
```



Carrying out the association tests

The association tests for imputed SNPs can be carried out using the function `single.snp.tests`.

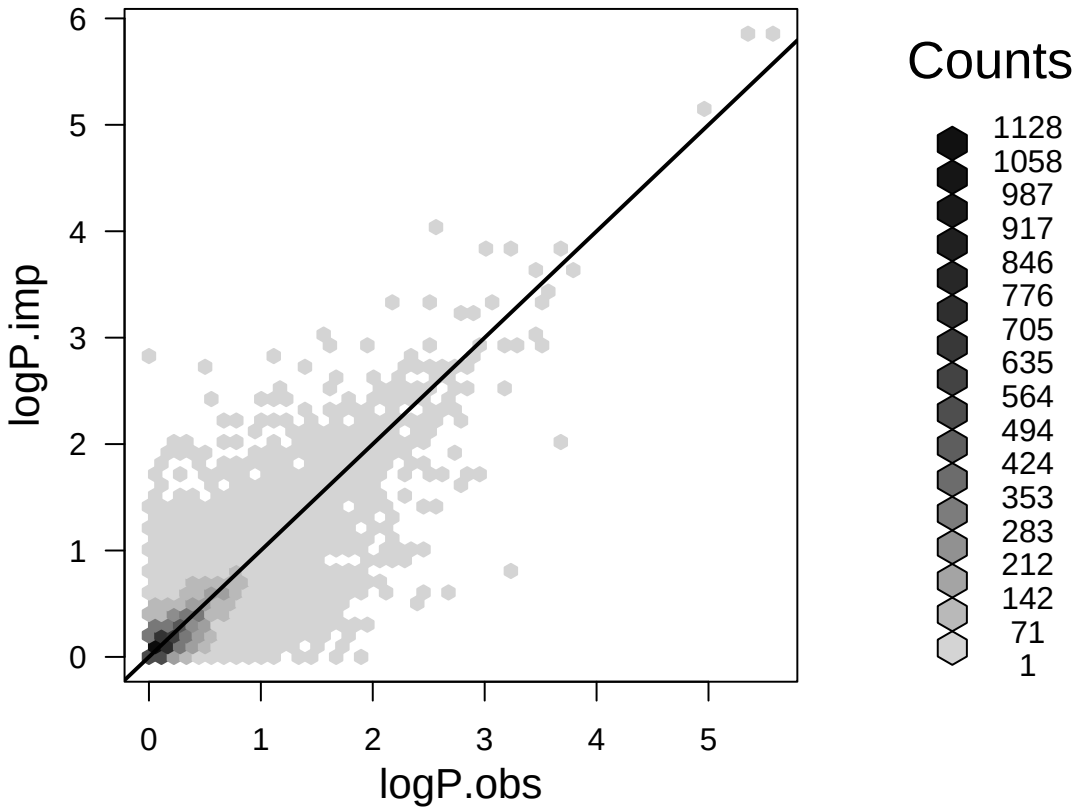
```
> imp <- single.snp.tests(cc, stratum, data = subject.support,
+     snp.data = present, rules = rules)
```

Using the observed data in the matrix `present` and the set of imputation rules stored in `rules`, the above command imputes each of the imputed SNPs, carries out 1- and 2-df single tests for association, returns the results in the object `imp`. To see how successful imputation has been, we can carry out the same tests using the *true* data in `missing`:

```
> obs <- single.snp.tests(cc, stratum, data = subject.support,
+     snp.data = missing)
```

The next commands extract the p-values for the 1-df tests, using both the imputed and the true "missing" data, and plot one against the other (using the `hexbin` plotting package for clarity):

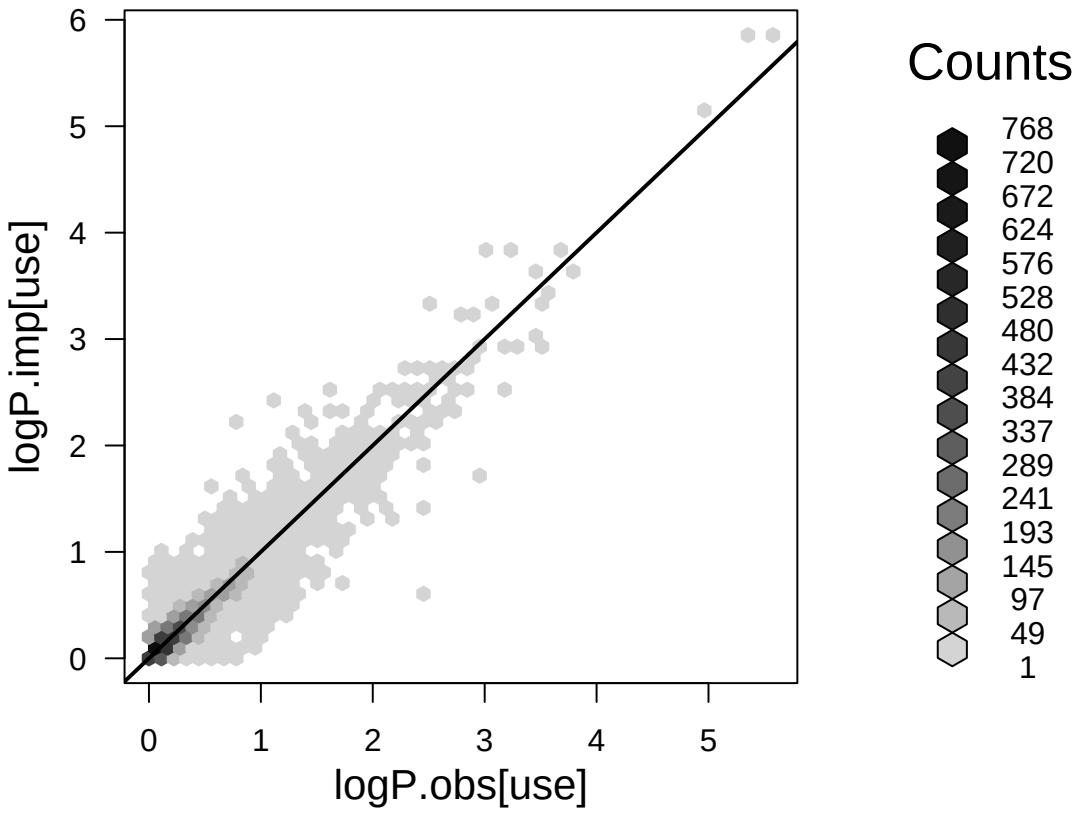
```
> logP.imp <- -log10(p.value(imp, df = 1))
> logP.obs <- -log10(p.value(obs, df = 1))
> hb <- hexbin(logP.obs, logP.imp, xbin = 50)
> sp <- plot(hb)
> hexVP.abline(sp$plot.vp, 0, 1, col = "black")
```



As might be expected, the agreement is rather better if we only compare the results for SNPs that can be computed with high R^2. The R^2 value is extracted from the `rules` object, using the function `imputation.r2` and used to select a subset of rules:

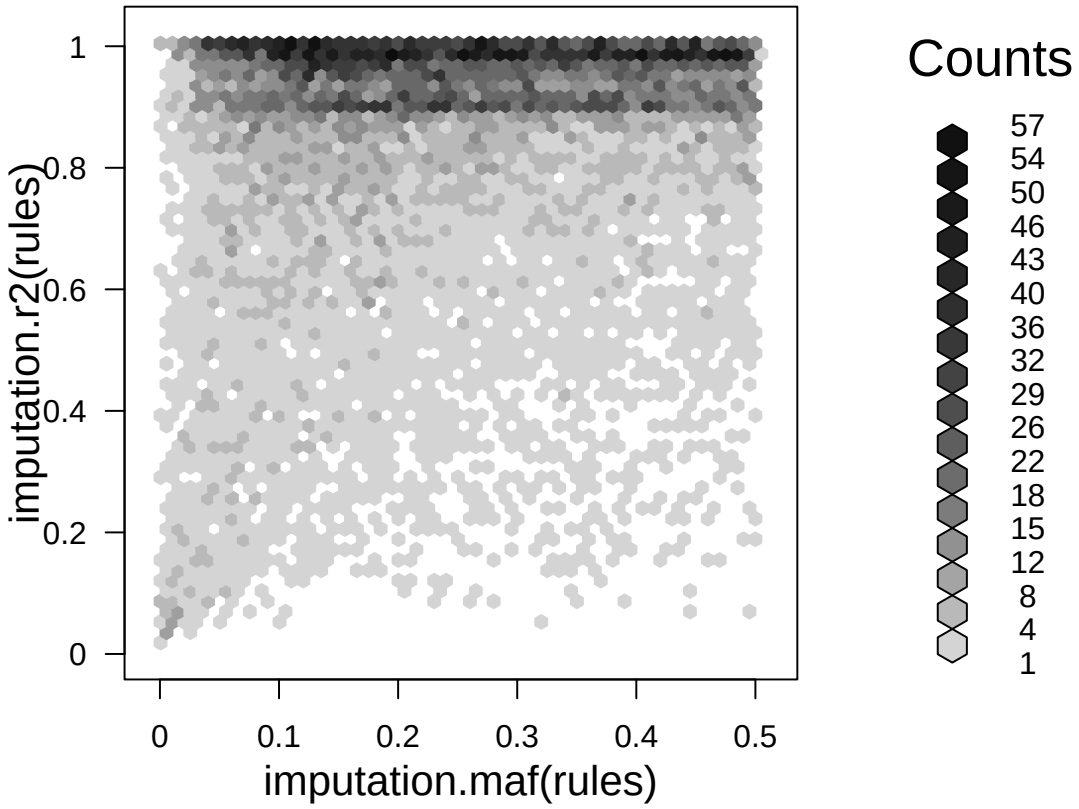
```
> use <- imputation.r2(rules) > 0.9
> hb <- hexbin(logP.obs[use], logP.imp[use], xbin = 50)
```

```
> sp <- plot(hb)
> hexVP.abline(sp$plot.vp, 0, 1, col = "black")
```



Similarly, the function `imputation.maf` can be used to extract the minor allele frequencies of the imputed SNP from the `rules` object. Note that there is a tendency for SNPs with a high minor allele frequency to be imputed rather more successfully:

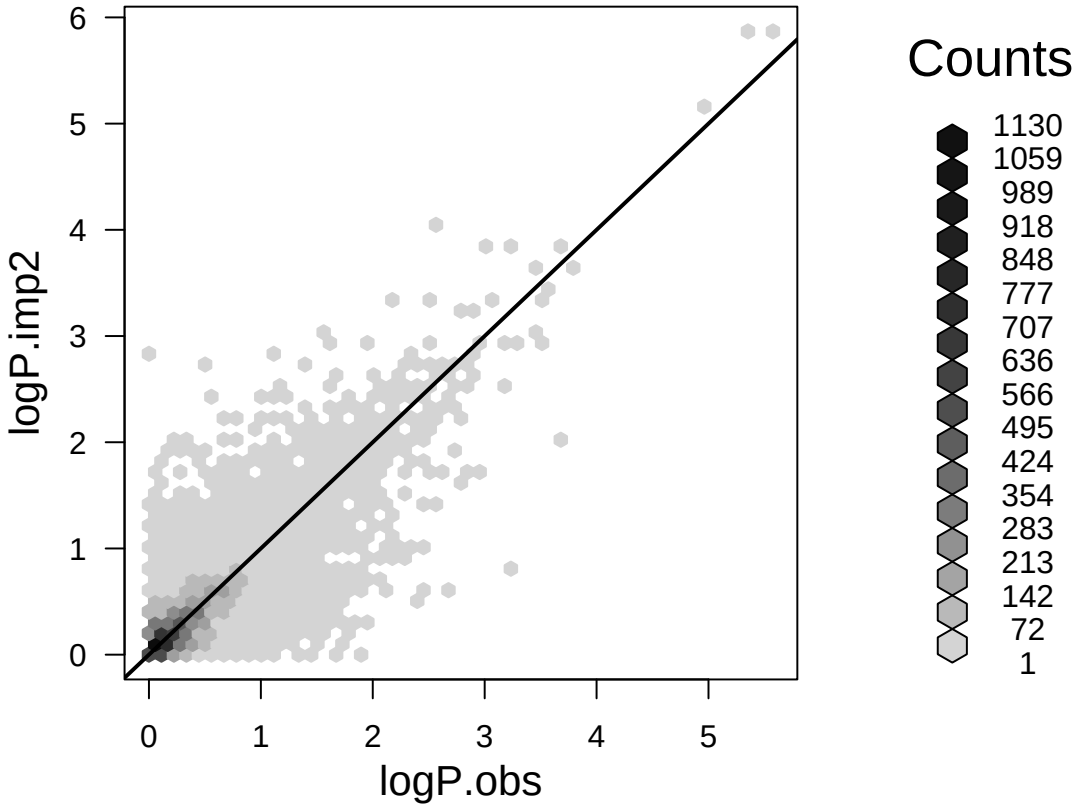
```
> hb <- hexbin(imputation.maf(rules), imputation.r2(rules), xbin = 50)
> sp <- plot(hb)
```



The function `snp.rhs.glm` also allows testing imputed SNPs. In its simplest form, it can be used to calculate essentially the same tests as carried out with `single.snp.tests`[1] (although, being a more flexible function, this will run somewhat slower). The next commands recalculate the 1 df tests for the imputed SNPs using `snp.rhs.tests`, and plot the results against those obtained when values are observed.

```
> imp2 <- snp.rhs.tests(cc ~ strata(stratum), family = "binomial",
+     data = subject.support, snp.data = present, rules = rules)
> logP.imp2 <- -log10(p.value(imp2))
> hb <- hexbin(logP.obs, logP.imp2, xbin = 50)
> sp <- plot(hb)
> hexVP.abline(sp$plot.vp, 0, 1, col = "black")
```

[1] There is a small discrepancy, of the order of $(N-1) : N$



Meta-analysis

As stated at the beginning of this document, one of the main reasons that we need imputation is to perform meta-analyses which bring together data from genome-wide studies which use different platforms. The `snpMatrix` package includes a number of tools to facilitate this. All the tests implemented in `snpMatrix` are "score" tests. In the 1 df case we calculate a score defined by the first derivative of the log likelihood function with respect to the association parameter of interest at the parameter value corresponding to the null hypothesis of no association. Denote this by U. We also calculate an estimate of its variance, also under the null hypothesis — V say. Then U^2/V provides the chi-squared test on 1 df. This procedure extends easily to meta-analysis; given two independent studies of the same hypothesis, we simply add together to two values of U and the two values of V, and then calculate U^2/V as before. These ideas also extend naturally to tests of several parameters (2 or more df tests)

In `snpMatrix`, the statistical testing functions can be called with the option `score=TRUE`, causing an extended object to be saved. The extended object con-

tains the U and V values, thus allowing later combination of the evidence from different studies. We shall first see what sort of object we have calculated using `single.snp.tests` *without* the `score=TRUE` argument.

```
> class(imp)

[1] "snp.tests.single"
attr(,"package")
[1] "snpMatrix"

> class(obs)

[1] "snp.tests.single"
attr(,"package")
[1] "snpMatrix"
```

We'll now recalculate these objects, saving the score information

```
> obs <- single.snp.tests(cc, stratum, data = subject.support,
+     snp.data = missing, score = TRUE)
> imp <- single.snp.tests(cc, stratum, data = subject.support,
+     snp.data = present, rules = rules, score = TRUE)
> class(obs)

[1] "snp.tests.single.score"
attr(,"package")
[1] "snpMatrix"

> class(imp)

[1] "snp.tests.single.score"
attr(,"package")
[1] "snpMatrix"
```

You will see that extended objects have been returned. These extended objects behave in the same way as the original objects, so that the same functions can be used to extract chi-squared values, p-values etc., but several additional functions, or methods, are now available. Chief amongst these is `pool`, which combines evidence across independent studies as described at the beginning of this section. Although `obs` and `imp` are *not* from independent studies, so that the resulting test would not be valid, we can use them to demonstrate this:

```
> both <- pool(obs, imp)
> class(both)

[1] "snp.tests.single"
attr(,"package")
[1] "snpMatrix"

> both[1:5]
```

```
              N     N.r2 Chi.squared.1.df Chi.squared.2.df      P.1df     P.2df
rs7909677  1943 1612.010       0.12766846        0.1280266 0.72086181 0.9379925
rs12773042 1942 1579.626       0.04845484        0.5396816 0.82577411 0.7635010
rs11253563 1979 1901.302       3.07211895        3.6364413 0.07964560 0.1623143
rs4881552  1958 1876.450       1.17566466        1.3058297 0.27824036 0.5205263
rs10904596 1970 1883.313       3.39998067        3.4145323 0.06519718 0.1813609
```

Note that if we wished at some later stage to combine the results in `both` with a further study, we would also need to specify `score=TRUE` in the call to `pool`:

```
> both <- pool(obs, imp, score = TRUE)
> class(both)

[1] "snp.tests.single.score"
attr(,"package")
[1] "snpMatrix"
```

Another reason to save the score statistics is that this allows us to investigate the *direction* of findings. These can be extracted from the extended objects using the function `effect.sign`. For example, this command tabulates the signs of the associations in `obs`:

```
> table(effect.sign(obs))

  -1    0    1
7085    3 7163
```

Reversal of sign can be the explanation of a puzzling phenomenon when two studies give significant results individually, but no significant association when pooled. Although it is not impossible that such results are genuine, a more usual explanation is that the two alleles have been differently coded in the two studies: allele 1 in the first study is allele 2 in the second study and vice versa. To allow for this `snpMatrix` provides the `switch.alleles` function, which reverses the coding of specified SNPs. It can be applied to `snp.matrix` objects but, because allele switches are often discovered quite late on in the analysis and recoding the original data matrices could have unforeseen consequences, the `switch.alleles` function can also be applied to the extended test output objects. This modifies the saved scores *as if* the allele coding had been switched in the original data. The use of this is demonstrated below.

```
> effect.sign(obs)[1:6]

 rs7909677 rs12773042 rs11253563  rs4881552 rs10904596  rs4880781
        -1         -1          1         -1         -1          1

> sw.obs <- switch.alleles(obs, c("rs12773042", "rs10904596"))
> class(sw.obs)

[1] "snp.tests.single.score"
attr(,"package")
[1] "snpMatrix"

> effect.sign(sw.obs)[1:6]

 rs7909677 rs12773042 rs11253563  rs4881552 rs10904596  rs4880781
        -1          1          1         -1          1          1
```