Imputed SNP analyses and meta-analysis with chopsticks

David Clayton

April 14, 2011

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

We shall sample 200 subjects in our fictitious study as the training data set, select alternate SNPs to be potentially missing or present in the target dataset, and split the training set into two parts accoordingly:

```
> training <- sample(1000, 200)
> in.target <- seq(1, ncol(snps.10), 2)
> missing <- snps.10[training, -in.target]
> present <- snps.10[training, in.target]
> missing
```

```
A snp.matrix with  200 rows and  14250 columns
Row names:  jpt.79 ... jpt.698
Col names:  rs7093061 ... rs7899159
```

```
> present
```

```
A snp.matrix with  200 rows and  14251 columns
Row names:  jpt.79 ... jpt.698
Col names:  rs7909677 ... rs12218790
```

Thus the training dataset consists of the objects `missing` and `present`. The target dataset holds a subset of the SNPs for the remaining 800 subjects.

```
> target <- snps.10[-training, in.target]
> target
```

```
A snp.matrix with  800 rows and  14251 columns
Row names:  jpt.869 ... ceu.464
Col names:  rs7909677 ... rs12218790
```

But, in order to see how successful we have been with imputation, we will also save the SNPs we have removed from the target dataset

```
> lost <- snps.10[-training, -in.target]
> lost
```

```
A snp.matrix with  800 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$position[-in.target]
> pos.pres <- snp.support$position[in.target]
```

Calculating the imputation rules

The next step is to calculate a set of rules which for imputing the `missing` SNPs from the `present` SNPs. This is carried out by the function `snp.imputation`[1]:

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

[1]Sometimes this command generates a warning message concerning the maximum number of EM iterations. If this only concerns a small proportion of the SNPs to be imputed it can be ignored.

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 0.95, or after including 4 SNPs in the regression, or until R^2 is not improved by at least 0.05. The figure 50 is the default value of the `try` argument of the function, while the values 0.95, 4 and 0.05 together make up the default value of the `stopping` argument. Where this regression equation provides adequate prediction, the corresponding element of `rules` contains the regression coefficients together with the R^2 achieved and the minor allele frequency of the target SNP. Where prediction does not achieve a target R^2, phased haplotype frequencies are estimated for the predictor SNPs plus the target SNP and a prediction rule based on these is evaluated. When the gain in R^2 exceeds a threshold, this haplotype-based rule is saved in preference to the regression based rule. The R^2 target and threshold which control this process are supplied in the argument `use.haps`.

A short listing of the first 10 rules follows:

```
> rules[1:10]

rs7093061  ~  rs11253563+rs754034+rs12573723 (MAF = 0.2272727, R-squared = 0.9079632)
rs7475011  ~  rs4881552*rs754034*rs10903844*rs12357593 (MAF = 0.3705584, R-squared = 0.8
rs4881551  ~  rs4881552+rs11253563 (MAF = 0.3838384, R-squared = 0.993976)
rs4880750  ~  rs2379080+rs4881552+rs11253563+rs7910845 (MAF = 0.2831633, R-squared = 0.8
rs7081782  ~  rs1476129+rs2448365 (MAF = 0.0725, R-squared = 0.9053637)
rs7898275  ~  rs1545003 (MAF = 0.06632653, R-squared = 1)
rs4880809  ~  rs6560730+rs17221309 (MAF = 0.2462312, R-squared = 0.9271854)
rs4390277  ~  rs1476129+rs2448365+rs9329280+rs10903844 (MAF = 0.075, R-squared = 0.89771
rs9419496  ~  rs7919436 (MAF = 0.2578125, R-squared = 0.9930367)
rs9419498  ~  rs4880517 (MAF = 0.05357143, R-squared = 0.9760987)
```

The rules are also selectable by SNP for detailed examination:

```
> rules[c("rs7898275", "rs9419496")]

rs7898275  ~  rs1545003 (MAF = 0.06632653, R-squared = 1)
rs9419496  ~  rs7919436 (MAF = 0.2578125, R-squared = 0.9930367)
```

Regression-based rules are shown with a `+` symbol separating predictor SNPs, while haplotype-based rules are shown with a `*` separator. A summary table of all the 14,251 rules is generated by

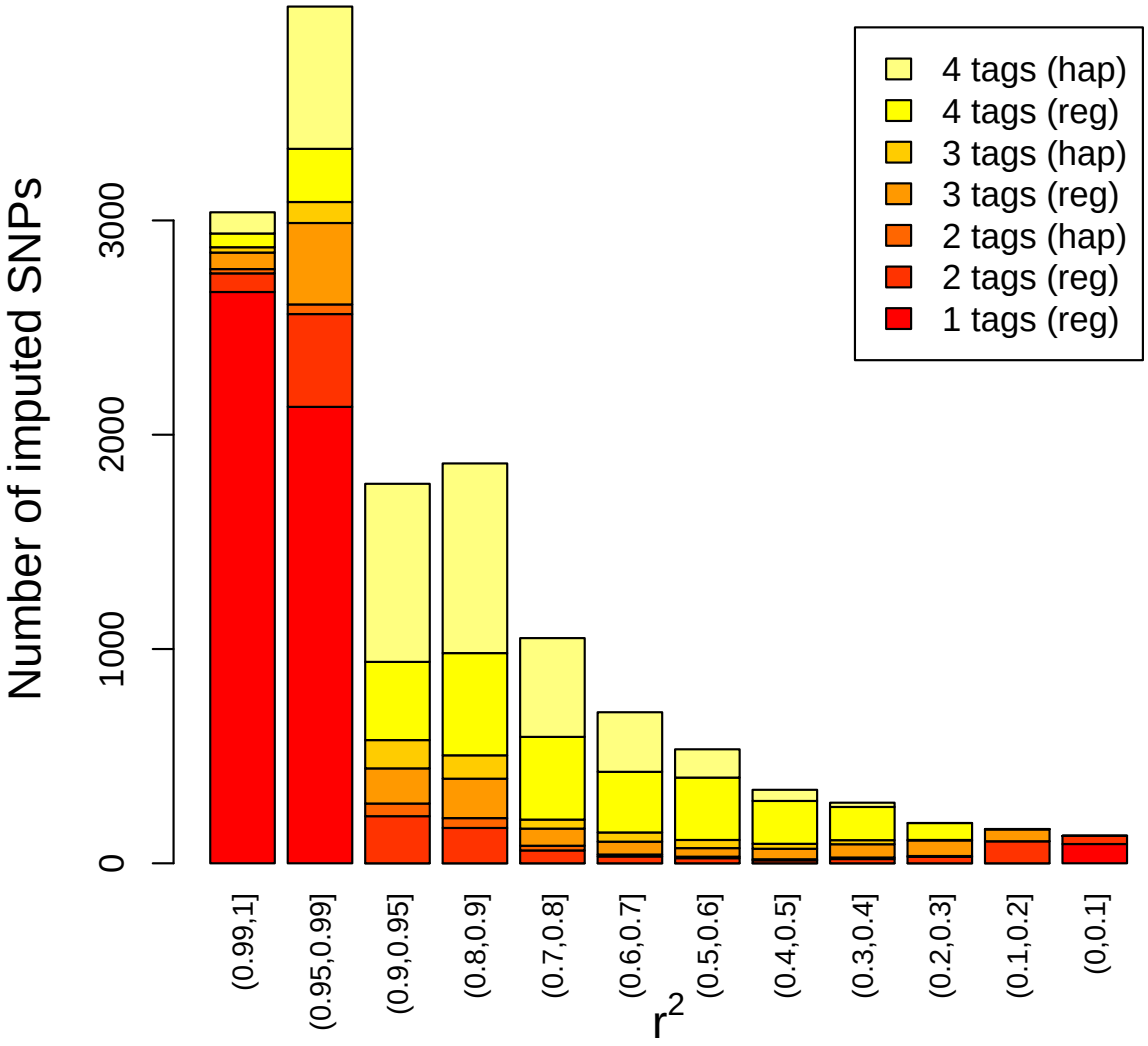
```
> summary(rules)

             SNPs used
R-squared     1 tags (reg) 2 tags (reg) 2 tags (hap) 3 tags (reg) 3 tags (hap)
```

```
  (0,0.1]              90           38            0            0            0
  (0.1,0.2]             0          103            0           56            0
  (0.2,0.3]             0           30            4           73            2
  (0.3,0.4]             0           20            7           62           19
  (0.4,0.5]             0           14            5           49           23
  (0.5,0.6]             0           24            7           40           38
  (0.6,0.7]             0           32            9           60           43
  (0.7,0.8]             0           60           22           80           42
  (0.8,0.9]             0          165           46          184          109
  (0.9,0.95]            0          220           59          164          132
  (0.95,0.99]        2130          433           45          380           98
  (0.99,1]           2666           87           20           77           25
  <NA>                  0            0            0            0            0
             SNPs used
R-squared     4 tags (reg) 4 tags (hap) <NA>
  (0,0.1]                0            0    0
  (0.1,0.2]              1            0    0
  (0.2,0.3]             79            0    0
  (0.3,0.4]            155           20    0
  (0.4,0.5]            200           52    0
  (0.5,0.6]            291          132    0
  (0.6,0.7]            283          278    0
  (0.7,0.8]            386          461    0
  (0.8,0.9]            477          885    0
  (0.9,0.95]           365          831    0
  (0.95,0.99]          248          664    0
  (0.99,1]              64           99    0
  <NA>                   0            0  187
```

Columns represent the number of SNPs and the type of rule, while rows represent grouping on R^2. The last column (headed `<NA>`) represents SNPs for which an imputation rule could not be computed, either because they were monomorphic or because there was insufficient data (as determined by the `minA` optional argument in the call to `snp.imputation`). 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 = target, rules = rules)
```

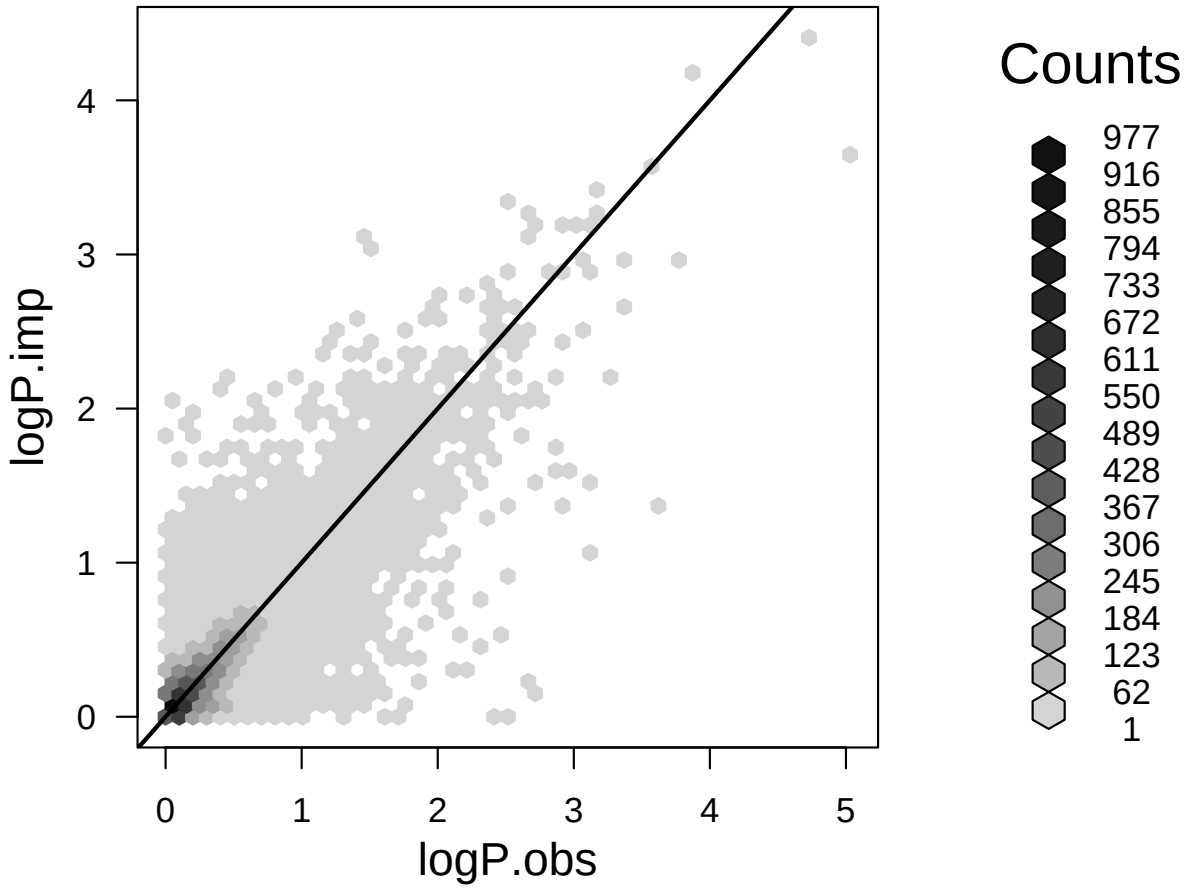
Using the observed data in the matrix `present` and the set of imputation rules stored in `rules`, the above command imputes each of the imputed SNPs, carries out 1- and 2-df single

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 = lost)
```

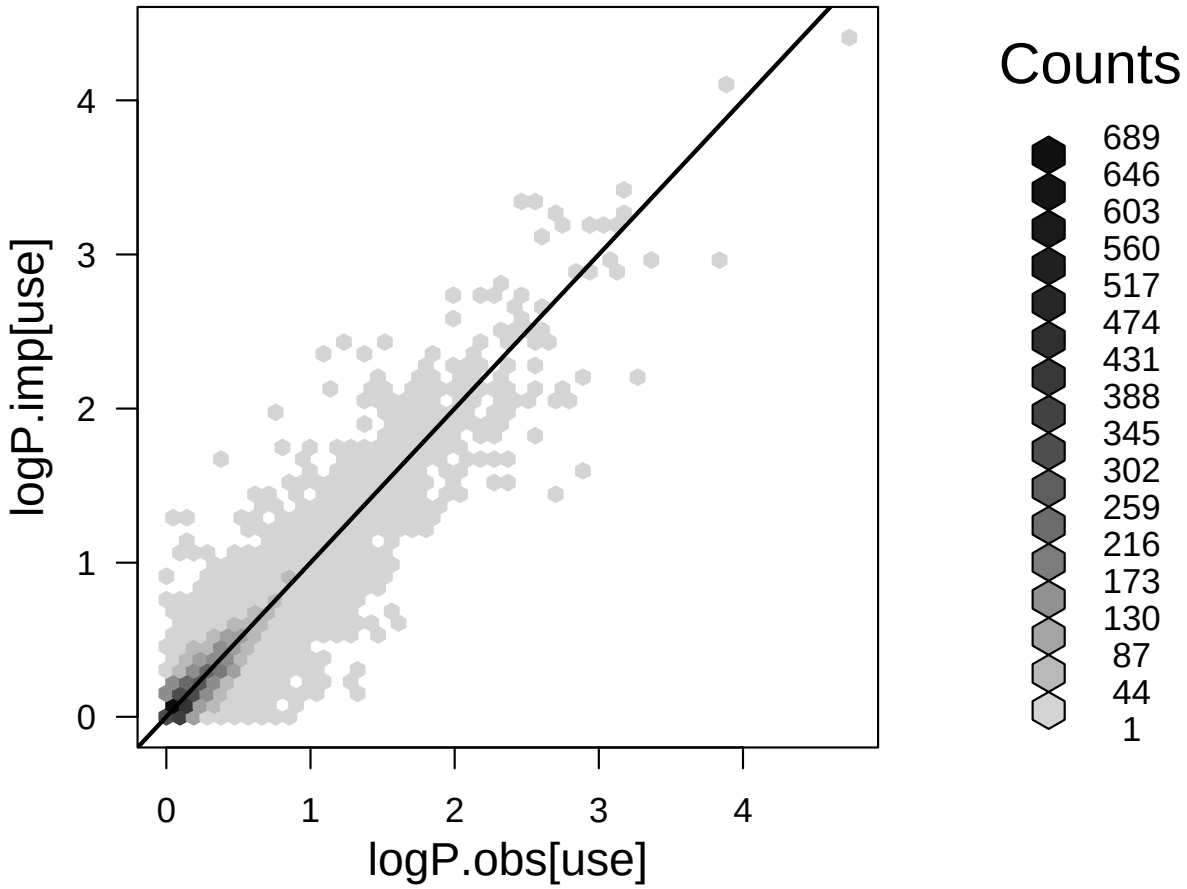
The next commands extract the p-values for the 1-df tests, using both the imputed and the true "missing" data, and plot one against the other (using the `hexbin` plotting package for clarity):

```
> 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")
```



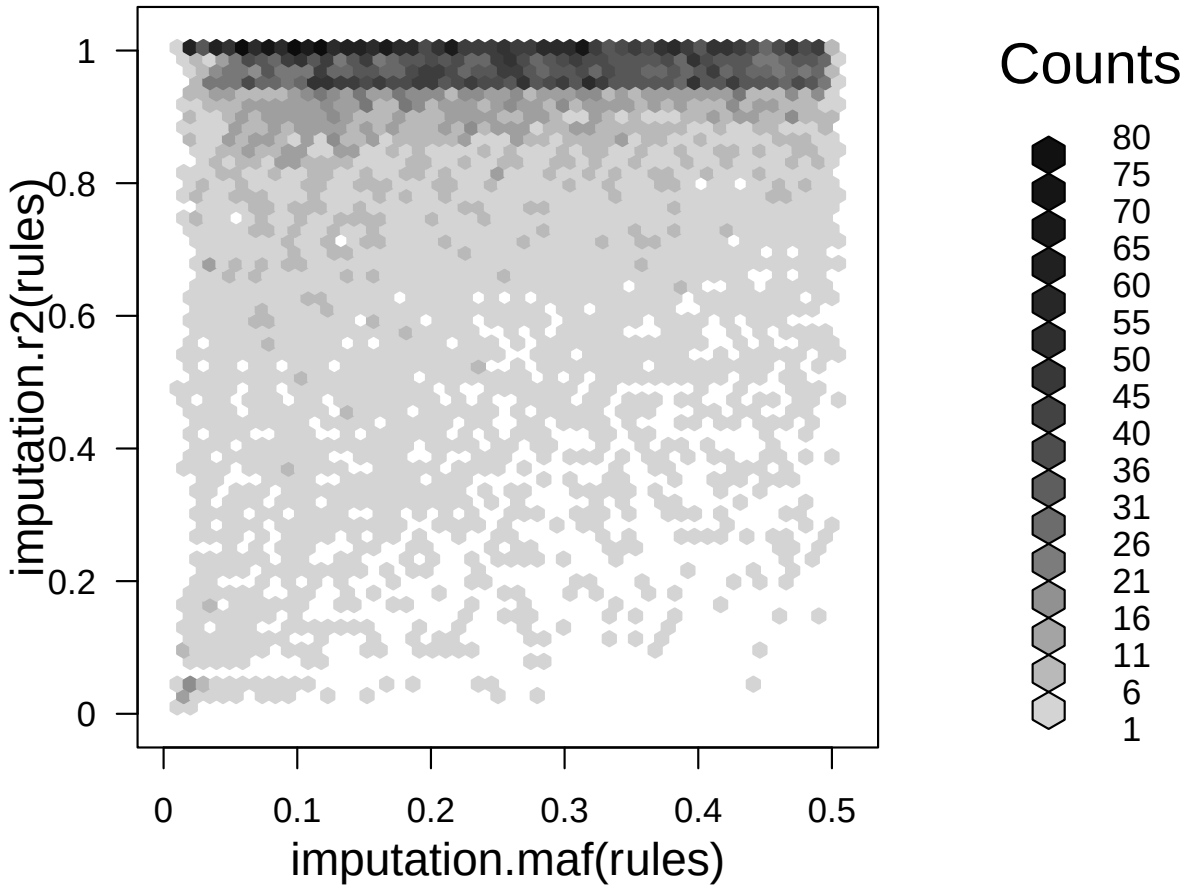
As might be expected, the agreement is rather better if we only compare the results for SNPs that can be computed with high R^2. The R^2 value is extracted from the `rules` object, using the function `imputation.r2` and used to select a subset of rules:

```
> 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)
```

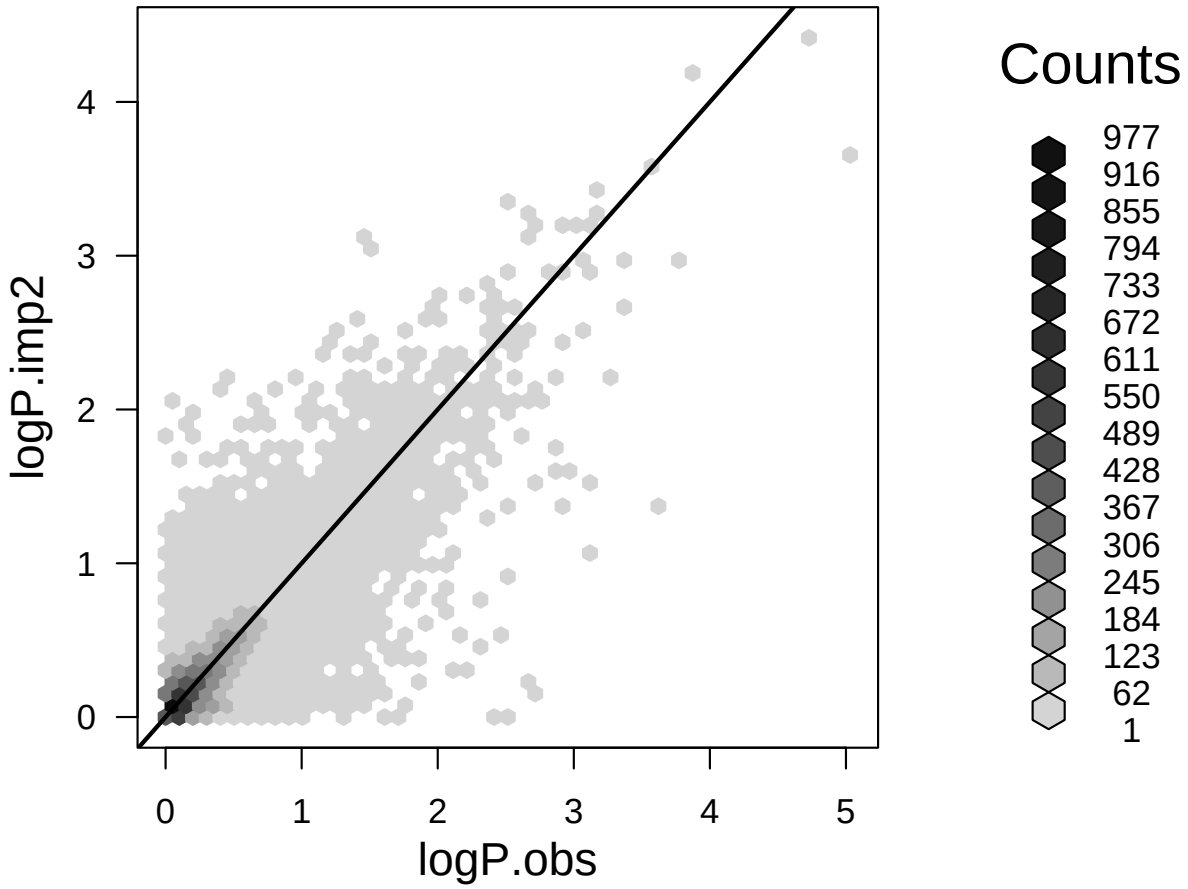


The function `snp.rhs.glm` also allows testing imputed SNPs. In its simplest form, it can be used to calculate essentially the same tests as carried out with `single.snp.tests`[2] (although, being a more flexible function, this will run somewhat slower). The next commands recalculate the 1 df tests for 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 = target, rules = rules)
> logP.imp2 <- -log10(p.value(imp2))
```

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

```
> hb <- hexbin(logP.obs, logP.imp2, xbin = 50)
> sp <- plot(hb)
> hexVP.abline(sp$plot.vp, 0, 1, col = "black")
```



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 `chopsticks` package includes a number of tools to facilitate this. All

the tests implemented in `chopsticks` 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 the 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 `chopsticks`, the statistical testing functions can be called with the option `score=TRUE`, causing an extended object to be saved. The extended object contains 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 previously using `single.snp.tests` *without* the `score=TRUE` argument.

```
> class(imp)

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

This object contains the imputed SNP tests in our target set. However, these SNPs were observed in our training set, so we can test them. We will also recalculate the imputed tests. In both cases we will save 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 = target, rules = rules, score = TRUE)
```

The extended objects have been returned:

```
> class(obs)

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

> class(imp)

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

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] "chopsticks"

> both[1:5]

            N     N.r2 Chi.squared.1.df Chi.squared.2.df     P.1df     P.2df
rs7093061 975 903.4874       1.84307517        1.9050217 0.1745909 0.3857712
rs7475011 997 903.1220       0.57583838        0.5834111 0.4479482 0.7469884
rs4881551 977 972.3073       0.07096543        0.1409142 0.7899363 0.9319677
rs4880750 958 848.0383       0.19973089        0.2728395 0.6549382 0.8724763
rs7081782 989 914.3320       0.88276176        1.2998059 0.3474464 0.5220965
```

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] "chopsticks"
```

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
7226   51 6973
```

In this table, -1 corresponds to tests in which effect sizes were negative (corresponding to an odds ratio less than one), while +1 indicates positive effect sizes (odds ratio greater than one). Zero sign indicates that the effect was `NA` (for example because the SNP was monomorphic).

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 coded differently in the two studies: allele 1 in the first study is allele 2 in the second study and vice versa. To allow for this, `chopsticks` 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]

rs7093061 rs7475011 rs4881551 rs4880750 rs7081782 rs7898275
       -1        -1        -1        -1         1         1

> sw.obs <- switch.alleles(obs, c("rs7093061", "rs7475011"))
> class(sw.obs)

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

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

rs7093061 rs7475011 rs4881551 rs4880750 rs7081782 rs7898275
        1         1        -1        -1         1         1
```