Computing min P test - a gene region-level testing procedure - with the `minPtest` function using the example of simulated SNP data

Stefanie Hieke

November 26, 2012

1 Introduction

This vignette documents the use of the `minPtest` function to compute the min P test, a gene region-level testing procedure using simulated single nucleotide polymorphisms (SNP) data with known structure, generated by the `generateSNPs` function.

2 The minPtest package

The **minPtest** package was written to provide a gene region-level summary for each candidate gene using the min P test for genetic case-control studies. The min P test is a permutation-based method that can be based on different univariate tests per SNPs. The package brings together three different kinds of tests that are scattered over several **R** packages. Calculations of the p-values from permutation-based methods can be time-consuming for large data sets, therefore, the `minPtest` package integrates two different parallel computing packages to improve computation speed by parallel computing. The use of **minPtest** is illustrated through two simulated data sets generated by the function `generateSNPs` of the **minPtest** package.

3 Generation of the simulated data sets

To illustrate the computation of the gene region-level summary, min P test, for different scenarios and settings, we generate SNP data with the `generateSNPs`

function, which is included in the `minPtest` package. We start by loading the **minPtest** package.

```
> library(minPtest)
```

The function `generateSNPs` simulates a matrix consisting of `n` subjects and `snp.no` SNPs, two automatically generated covariates and matchset numbers. Note that `n` has to be specified as an argument in the `generateSNPs` function and the number of SNPs `snp.no` is derived from the specified arguments `gene.no`, `block.no` and `block.size` in the `generateSNPs` function. SNPs with genotypes coded by 0, 1 and 2 are simulated using the probability for neighborhood SNPs within a block `p.same`, the probability for neighborhood blocks within a gene `p.different` and the allele frequency `p.minor`. The argument `p.same` can either be a numeric value, see example 3.1 or a vector of length `block.size` where the first item is fixed as the value which would be selected for the probability for neighborhood blocks. In the latter case the argument `p.different` is ignored. The remaining items in the `p.same` vector specify the probability for each neighborhood SNP within the blocks, see example 3.2. The response is determined by a logistic regression model given the SNPs and the two covariates. The parameters (effect size) of the SNPs for the generation of case-control status are specified in the `SNPtoBETA` matrix, see 3.1 and 3.2.

3.1 First simulated data set scenario

We simulated a (rather) small genetic case-control data set. The example below illustrates the generation of a data set consisting of 100 observations, 200 SNPs with genotypes coded by 0, 1 and 2, two clinical covariates (continuous and binary) for adjustment or matching and the matchset numbers. The SNPs are located on 5 genes with 4 blocks per gene and block size 10, i.e. 10 SNPs per block yielding 40 SNPs per gene. First we will define the `SNPtoBETA` matrix in order to specify the SNPs and their effect size (offsets) which are explanatory for the response (case-control status). In this scenario, we select two SNPs located on G1 in two blocks with allele frequency 0.1 and moderate effect size, one SNPs located on G2 in one block with allele frequency 0.4 and high effect size, two SNPs located on G4 in two blocks with allele frequency 0.4 and moderate effect size and one SNP located on G5 in one block with allele frequency 0.1 and moderate effect size.

```
> SNP <- c(6,26,54,135,156,186)
> BETA <- c(0.9,0.7,1.5,0.5,0.6,0.8)
> SNPtoBETA <- matrix(c(SNP,BETA),ncol=2,nrow=6)
> colnames(SNPtoBETA) <- c("SNP.item","SNP.beta")
```

We simulate the data set with equal neighborhood probability (0.9) for each SNP within each block and equal probabilities for the neighborhood blocks (0.75) within a gene. We set a seed for reproducibility.

```
> set.seed(1007)
> sim.ex1 <- generateSNPs(n=100,gene.no=5,block.no=4,block.size=10,
                          p.same=0.9,p.different=0.75,
                          p.minor=c(0.1,0.4,0.1,0.4),
                          n.sample=80,SNPtoBETA=SNPtoBETA)
```

One output of this function is the simulated data matrix `sim.data` with one row for each observation containing response values, simulated SNP values (coded by 0, 1 and 2.), a continuous and a binary covariate and a matchset number. The following command returns an excerpt of the data set and should not be called if the data set is to large.

```
> # head(sim.ex1$sim.data)
```

The function also returns a list of outputs which can be directly used as input for the `minPtest` function where `y` is a numeric response vector coded with 0 (coding for controls) and 1 (coding for cases), `x` is a numeric matrix containing the simulated SNP data with genotypes coded by 0, 1 and 2, `SNPtoGene` is a matrix comprising the SNP names and the names of the genes on which the SNPs are located, `cov` is a matrix containing a continuous and a binary clinical covariate and `matchset` is a numeric vector containing the matching numbers. The `print` function displays brief information on the simulated data set and the number of SNPs.

```
> sim.ex1
Call:  generateSNPs(n = 100, gene.no = 5, block.no = 4,
block.size = 10, p.same = 0.9, p.different = 0.75,
p.minor = c(0.1, 0.4, 0.1, 0.4), n.sample = 80, SNPtoBETA = SNPtoBETA)

Simulated data set containing 200 SNPs, two matching covariates and
a matchset column (containing the matchset numbers).
Output y, x, SNPtoGene, cov and matchset can directly be used
for the minPtest function.
```

3.2 Second simulated data set scenario

The example below uses a data set as in 3.1 except for the probability for the neighborhood SNPs within the blocks, the SNP positions and their effect sizes for

the generation of the case-control status. A break within the first block is specified in each gene. Therefore, we change the probability for the neighborhood SNPs of the SNP-position 6 from 0.9 to 0.5 and retain equal neighborhood probability for each block as in 3.1 by fixing the value 0.75 as in 3.1 at the first entry of the `p.same` vector and retaining the default value `NULL` of `p.different`. Furthermore, we have to define the `SNPtoBETA` matrix in order to specify the SNPs and their effect size (offsets) which are explanatory for the response (case-control status). In this scenario, we select two SNPs located on G1 in two blocks with allele frequency 0.1 or 0.4 and high effect size, one SNPs located on G3 in one block with allele frequency 0.4 and moderate effect size, two SNPs located on G4 in two blocks with allele frequency 0.1 or 0.4, respectively, and moderate effect size and two SNPs located on G5 in two blocks with allele frequency 0.1 and moderate effect size.

```
> p.same <- rep(c(0.75,rep(0.9,9)),4)
> p.same[6] <- 0.5
> SNP <- c(7,15,96,145,157,164,185)
> BETA <- c(1.5,1.0,0.5,0.8,0.4,0.6,0.8)
> SNPtoBETA <- matrix(c(SNP,BETA),ncol=2,nrow=7)
> colnames(SNPtoBETA) <- c("SNP.item","SNP.beta")

> set.seed(2006)
> sim.ex2 <- generateSNPs(n=100,gene.no=5,block.no=4,block.size=10,
                          p.same=p.same,p.minor=c(0.1,0.4,0.1,0.4),
                          n.sample=80,SNPtoBETA=SNPtoBETA)
```

This function returns a similar list as described in 3.1. The following command returns an excerpt of the data set and should not be called if the data set is to large.

```
> # head(sim.ex2$sim.data)
```

4 Candidate gene analysis using the min P test

The main focus of the **minPtest** package is the computation of the permutation based p-values for candidate genes using the min P test. The gene region-level summary, as the min P test, assesses the statistical significance of the smallest p-trend within each gene region comparing cases and controls. Inference is based on the permutation distribution of the minimum of the ordered p-values from the marginal test of each SNP. The permutation method can be based on different univariate tests per SNP and the `minPtest` function brings together three different

kinds of tests to compute such p-values.
In order to illustrate the computation of the gene region-level testing procedure using unconditional and conditional logistic regression, respectively, to compute marginal and permuted trend p-values, we use the simulated data sets of 3.1 and 3.2. An example to compute the gene region-level summary using an Cochran Armitage Trend Test is given at the help page of the `minPtest` function. As the computation of the p-values from permutation-based methods can be time-consuming, we use the **multicore** and the **snowfall** package to obtain acceleration by parallel computing in the examples 4.1 or 4.2. An example for a sequential application is given on the help page of the `minPtest` function.

4.1 Computing the min P test using unconditional logistic regression and multicore

We start by accessing the simulated data set of 3.1. A short explanation might be useful before calling the `minPtest` function. The **minPtest** package brings together three different kinds of tests which are scattered over several **R** packages, and the `minPtest` function automatically selects the most appropriate test for the study design at hand. Trend p-values are computed using Cochran Armitage Trend Test including only a response vector, a SNP matrix and a matrix comprising the SNP names and the corresponding gene names on which the SNPs are located, see the help page of the `minPtest` function. In the example below, additional to the response vector, the SNP matrix and the matrix comprising SNP and the corresponding gene names generated in `sim.ex1` (3.1), we specify a formula to compute the trend p-values using the unconditional logistic regression. Note that no matchset vector is needed. There are two possibilities to specify the formula. First, if no covariates are used for adjustment in the unconditional logistic regression, the formula has to be written as `y`$\sim$1 without specifying the covariate matrix `cov`. Second, if covariates other than SNPs are used for adjustment in the unconditional logistic regression, the formula has to be written by the response vector `y` on the left of a $\sim$ operator and the clinical covariates on the right. In addition, a covariate matrix has to be specified. In the example below we use a continuous and a binary covariate simulated in 3.1 for adjustment. We run the call of the `minPtest` function on a multicore computer with 4 cores, in order to obtain acceleration by parallel computing, by setting option `multicore=4` which requires the installation of the **multicore** package, as the `minPtest` call can take some time. Note, **multicore** is currently not available on Microsoft Windows. An alternative to `multicore` for Microsoft Windows system users is the **snowfall** package, see 4.2. We set a seed to generate `seed1` of length `permutation` to guarantee reproducibility of the results even if running in parallel and for different

numbers of parallel processes.

```
> set.seed(10)
> seed1 <- sample(1:1e7,size=1000)

> minPtest.object1 <- minPtest(y=sim.ex1$y, x=sim.ex1$x,
                               SNPtoGene=sim.ex1$SNPtoGene,
                               formula=y~cov.continuous+cov.binary,
                               cov=sim.ex1$cov, multicore=4, seed=seed1)

> minPtest.object1

Used method: unconditional logistic regression (glm) for 100 subjects
Call:  minPtest(y = sim.ex1$y, x = sim.ex1$x,
SNPtoGene = sim.ex1$SNPtoGene, formula = y ~ cov.continuous + cov.binary,
cov = sim.ex1$cov, seed = seed1, multicore = 4)

Number of genes: 5
Number of SNPs: 200
Number of missings in the SNP matrix: 0
Number of permutations: 1000
```

Above, the display returned by the `print` function can be seen. The method used for the computation of the marginal and the permuted trend p-values, the number of the subjects, the number of the genes, the number of the SNPs used for the computation, the number of the missings in the SNPs and the number of the permutations used to compute the the permuted distribution of the minimum of the ordered p-values from the marginal test of each SNP are printed.
The `minPtest` function returns a list including a matrix of permutation-based p-values from min P test for each candidate gene, a matrix of corrected permutation-based p-values via Bonferroni correction method (default) for each candidate gene, a matrix of marginal trend p-values for each SNP from the original data set, a matrix of corrected marginal trend p-values via Bonferroni correction method (default) for each SNP from the original data set, a matrix of permuted trend p-values for each SNP in each permutation step, etc.
The main output of the `minPtest` call is the matrix of permutation-based p-values of the min P test for each candidate gene.

```
> minPtest.object1$minp
    minP
G1 0.301
G2 0.001
G3 0.328
G4 0.228
G5 0.244
```

More detailed information is provided by a `summary` function.

```
> summary(minPtest.object1,sign.SNP=TRUE)

p-values:
  Gene  minP gene.p.adjust   SNP  snp.p.value snp.p.adjust
1   G2 0.001         0.005 SNP54 0.0001362140   0.02724279
```

Above the display provided by the `summary` function can be seen. The table shows the gene with the adjusted permutation-based p-value smaller than or equal to a threshold (default 0.05), the corresponding permutation-based p-value, the adjusted permutation-based p-value as well as the SNP located on this gene with adjusted marginal p-value smaller than or equal to the threshold, as `sign.SNP=TRUE`, with marginal p-value and adjusted marginal p-value. The `summary` function is used to obtain a brief overview of the significant genes, after the correction for multiple hypothesis testing (default Bonferroni correction), and the SNPs located on these genes. If `sign.SNP=TRUE`, the summary shows the SNPs located on the genes selected according to the threshold, with adjusted marginal p-values smaller or equal to the threshold. Otherwise all SNPs located on the genes chosen by the threshold are shown in the summary.
The `summary` function returns a list of the same length as the number of the selected genes by a threshold. Each item characterizes a gene selected according to a threshold i.e. if `level=1`, the length of the list equals to the number of genes included in the fit. Each gene item contains a list of data frames, a data frame for the permutation-based p-values and adjusted permutation-based p-values for this gene and a data frame for the marginal p-values and adjusted marginal p-values for the SNPs located on this gene, either SNPs selected by a threshold or all SNPs on this gene.

4.2 Computing the min P test using conditional logistic regression and snowfall

We start by accessing the simulated data set of 3.2. We sampled 100 subjects (50 cases and 50 controls) and 200 SNPs on 5 genes in `sim.ex2` (3.2) as in 3.1

except for the probability for the neighborhood SNPs within the blocks as we generated a break within the first block at SNP position 6 in each gene, see 3.2. In this example, we illustrate the computation of the trend p-values using conditional logistic regression which requires the installation of package **Epi**. The computation of the trend p-values using conditional logistic regression is automatically selected by the `minPtest` function investigating the following input from `sim.ex2` (3.2). As in 4.1, we include the response vector, the SNP matrix and a matrix comprising the SNP names and the gene names on which the SNPs are located, generated in `sim.ex2`. Compared to 4.1, we use the continuous and the binary covariate simulated in 3.2 as matching variables through the matchset vector `matchset` and do not include them in the covariate matrix `cov` for adjustment. Therefore, as no covariates are used for adjustment, the formula has to be written as $y \sim 1$ without specifying a covariate matrix. However, a matchset vector has to be specified. We set a seed to guarantee reproducibility of the results, even for different numbers of parallel processes, see 3.1. We run the call of the `minPtest` function on a compute cluster using 4 CPUs, to obtain acceleration by parallel computing by setting option `parallel=TRUE` which requires the installation of the **snowfall** package, as the `minPtest` call can take some time. Concerning parallelization on a compute cluster, i.e. with argument `parallel=TRUE`, there are two possibilities to run the `minPtest` call:

- Start **R** on a commandline with sfCluster and preferred options, e.g. number of cpus. The initialization function of package **snowfall**, `sfInit()`, has to be called before calling the `minPtest` function.
  sfCluster is a Unix tool for convenient management of **R** parallel processes. It is available at `www.imbi.uni-freiburg.de/parallel`, with detailed information.

```
> sfInit()
> minPtest.object2 <- minPtest(y=sim.ex2$y, x=sim.ex2$x,
                               SNPtoGene=sim.ex2$SNPtoGene,
                               formula=y~1, matchset=sim.ex2$matchset,
                               parallel=TRUE, seed=seed1)
```

- Use any other solutions supported by **snowfall**. Argument `parallel` has to be set to TRUE and number of cpus can be chosen in the `sfInit()` function.

```
> sfInit(parallel=TRUE,cpus=4)
> minPtest.object2 <- minPtest(y=sim.ex2$y, x=sim.ex2$x,
                               SNPtoGene=sim.ex2$SNPtoGene,
                               formula=y~1, matchset=sim.ex2$matchset,
                               parallel=TRUE,seed=seed1)
```

The latter is an alternative to the parallelization on a multicore computer with `multicore` for Microsoft Windows system users.
Independent of the chosen initialization function, the following display is provided by the `print` function.

```
> minPtest.object2

Used method: conditional logistic regression for 100 subjects
Call:  minPtest(y = sim.ex2$y, x = sim.ex2$x,
SNPtoGene = sim.ex2$SNPtoGene, formula = y ~ 1,
matchset = sim.ex2$matchset, seed = seed1, parallel = TRUE)

Number of genes: 5
Number of SNPs: 200
Number of missings in the SNP matrix: 0
Number of permutations: 1000
```

The next command extracts the matrix of permutation-based p-values of the min P test for each candidate gene.

```
> minPtest.object2$minp
    minP
G1 0.004
G2 0.345
G3 0.792
G4 0.270
G5 0.312
```

More information is provided by a `summary` function which returns a list of data frames for each candidate gene selected according to a threshold, see 4.1. The following display shows the list items for G1 containing two data frames, a data frame for the permutation-based p-value and the adjusted permutation-based p-value for G1 and a data frame for the marginal p-values and adjusted marginal p-values for the SNPs located on G1.

```
> summary(minPtest.object2)$G1
$gene.p.values
  Gene  minP gene.p.adjust
1   G1 0.004          0.02

$snp.p.values
     SNP snp.p_value snp.p.adjust
```

1	SNP15	0.002070020	0.4140041
2	SNP16	0.002526884	0.5053769
3	SNP19	0.003429856	0.6859712
4	SNP17	0.003728256	0.7456512
5	SNP1	0.371943971	1.0000000
6	SNP2	0.409689081	1.0000000
7	SNP3	0.409689081	1.0000000
8	SNP4	0.326128617	1.0000000
9	SNP5	0.211861133	1.0000000
10	SNP6	0.039743515	1.0000000
11	SNP7	0.020706071	1.0000000
12	SNP8	0.022520147	1.0000000
13	SNP9	0.037072000	1.0000000
14	SNP10	0.052586757	1.0000000
15	SNP11	0.006508902	1.0000000
16	SNP12	0.016151701	1.0000000
17	SNP13	0.029085604	1.0000000
18	SNP14	0.018493517	1.0000000
19	SNP18	0.006899747	1.0000000
20	SNP20	0.005824611	1.0000000
21	SNP21	0.009517110	1.0000000
22	SNP22	0.007810166	1.0000000
23	SNP23	0.041817448	1.0000000
24	SNP24	0.036156703	1.0000000
25	SNP25	0.028889167	1.0000000
26	SNP26	0.107261909	1.0000000
27	SNP27	0.082974190	1.0000000
28	SNP28	0.057018754	1.0000000
29	SNP29	0.031694448	1.0000000
30	SNP30	0.115757602	1.0000000
31	SNP31	0.724040250	1.0000000
32	SNP32	0.869453258	1.0000000
33	SNP33	0.872810958	1.0000000
34	SNP34	0.865809068	1.0000000
35	SNP35	1.000000000	1.0000000
36	SNP36	1.000000000	1.0000000
37	SNP37	0.647879273	1.0000000
38	SNP38	0.869443031	1.0000000
39	SNP39	0.875918977	1.0000000
40	SNP40	0.731854289	1.0000000

4.3 Plotting permutation-based p-values for each candidate gene and marginal p-values for the SNPs located on these genes

The `plot` function is used to present the information provided by `summary` graphically, i.e. to display the permutation-based p-values for each candidate gene and the marginal p-values for each SNP located on these genes in a graphical way. The function plots either $(-\log_{10})$ transformed permutation-based p-values for each gene or $(-\log_{10})$ transformed marginal p-values for each SNP in a basic scatterplot using the argument `type="gene"` (default) or `"SNP"`. The y-axis is $(-\log_{10})$ transformed to obtain a disposition as a Manhattan plot for the points of the marginal p-values of the SNPs. Furthermore, an alternative given by the `plot` function is to display the marginal p-values for each SNP and the transformed permutation-based p-values for each gene in a combined plot using the argument `type="both"`. Accordingly, $(-\log_{10})$ transformed marginal p-values for each SNP are plotted as points. In addition, horizontal lines of $(-\texttt{lambda} * \log_{10})$ transformed permutation-based p-values of each gene, covering all SNPs located on that gene, are plotted. The composed plot is indicated by two separated y-axes $(-\log_{10}(\texttt{psnp}))$ at left hand side and $(-\texttt{lambda} * \log_{10}(\texttt{minp}))$ at the right hand side). `psnp` is a matrix of marginal trend p-values and `minp` is a matrix of permutation-based p-values from `minPtest` object, see 4.1 and 4.2. `lambda` is used to scale the y-axis for the log-transformed permutation-based p-values. After the correction for multiple hypothesis testing (default Bonferroni correction) depending on the `level` (default 0.05) significant genes and SNPs are by default highlighted in red. I.e. not depending on the used type of plot, each gene or/and each SNP with permutation-based p-value or/and marginal p-value smaller than or equal to the `level` is highlighted in red.

4.3.1 Plotting permutation-based and marginal p-values from 4.1

The combined plot of the resulting $(-0.5 \cdot \log_{10})$ transformed permutation-based p-values for each candidate gene and the $(-\log_{10})$ transformed marginal p-values for each SNP located on the genes from 4.1 with a scaled y-axis for the permutation-based p-values is displayed using the `plot` function and argument `type="both"`.

```
> plot(minPtest.object1, type="both", lambda=0.5, gene.name=TRUE)
```

Figure 1 shows horizontal lines for each $(-0.5 \cdot \log_{10})$ transformed permutation-based p-value of the candidate genes and dots for each $(-\log_{10})$ transformed marginal p-value of the SNPs located on these genes. The horizontal line of gene G2 and a dot of a SNP (SNP54, see `summary` in 4.1) are highlighted in red, as

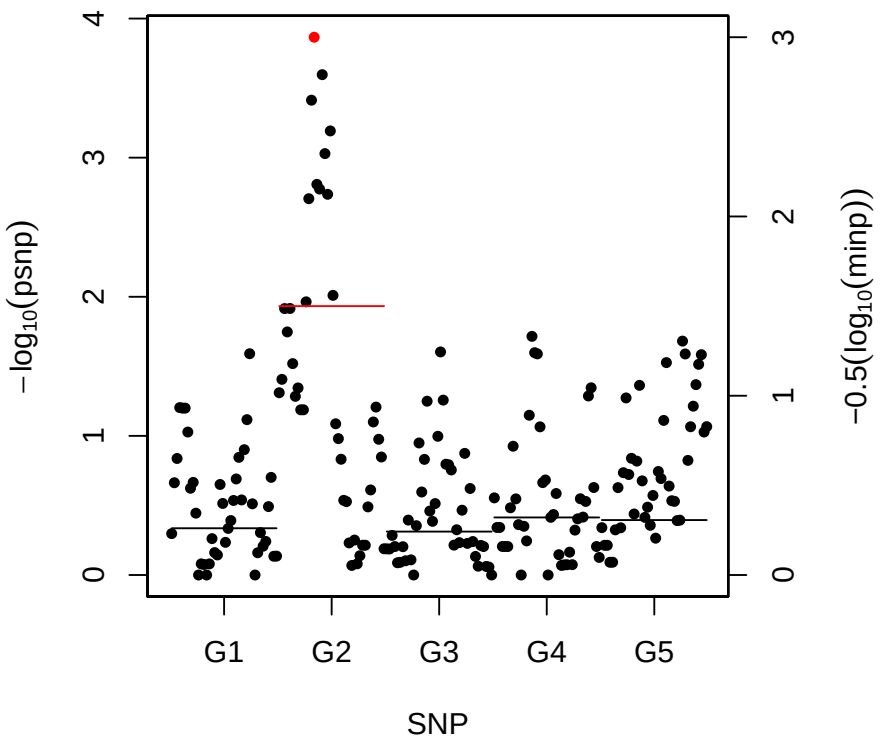

Figure 1: $(-0.5 \cdot \log_{10})$ transformed permutation-based p-values for each candidate gene and $(-\log_{10})$ transformed marginal p-values for each SNP located on these genes for 4.1. Dots: $(-\log_{10})$ transformed marginal p-values, lines: $(-0.5 \cdot \log_{10})$ transformed permutation-based p-values

their p-values are smaller than or equal to `level=0.05` after Bonferroni correction (default).
The `summary` and the `plot` illustrate that the gene region-level summary is mostly compatible with univariate statistical tests per SNP conducted separately over multiple loci. In both functions, G2 and SNP54 are highlighted which could be expected as we fixed an effect size of 1.5 for SNP54 which is located on G2.
It should be stressed that for real data the plot would usually contain a lot more genes and SNPs located on these genes. This would rather lead to a disposition as to a Manhattan plot for the points of the $(-\log_{10})$ transformed marginal p-values compared to Figure 1.

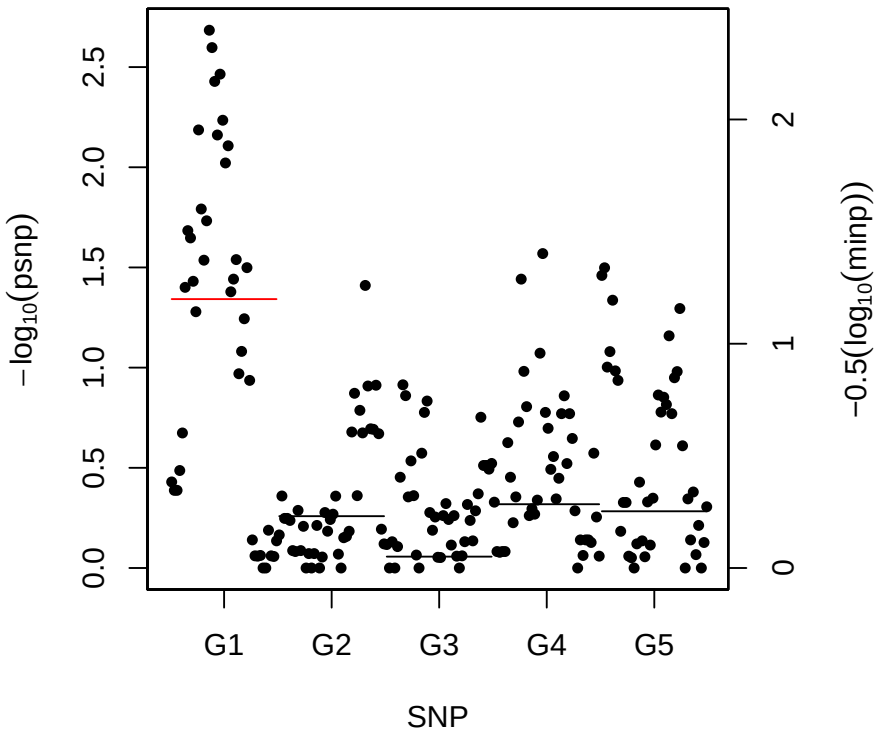

Figure 2: $(-0.5 \cdot \log_{10})$ transformed permutation-based p-values for each candidate gene and $(-\log_{10})$ transformed marginal p-values for each SNP located on these genes for 4.2. Dots: $(-\log_{10})$ transformed marginal p-values, lines: $(-0.5 \cdot \log_{10})$ transformed permutation-based p-values

4.3.2 Plotting permutation-based and marginal p-values from 4.2 with a break in the first block of each gene

The combined plot of the resulting $(-0.5 \cdot \log_{10})$ transformed permutation-based p-values for each candidate gene and the $(-\log_{10})$ transformed marginal p-values for each SNP located on these genes from 4.2 with a scaled y-axis for the permutation-based p-values is displayed using the `plot` function and argument `type="both"`. The difference to 4.1 is the break within the first block due to the modification of the probability for the neighborhood SNPs of the SNP-position 6 from 0.9 to 0.5 in each gene, see 3.2.

```
> plot(minPtest.object2, type="both", lambda=0.5, gene.name=TRUE)
```

Figure 2 shows horizontal lines for each $(-0.5 \cdot \log_{10})$ transformed permutation-based p-value of the candidate genes, as well as dots for each $(-\log_{10})$ transformed marginal p-value of the SNPs located on these genes. Compared to Figure 1, simply the horizontal line of gene G1 is highlighted in red as the Bonferroni corrected (default) permutation-based p-values is smaller than the default level and no Bonferroni corrected (default) marginal p-value is smaller or equal than the level, no dots of the SNP are highlighted in red, see also the list items from the `summary` function in 4.2.