Package 'RVS0.0'

Jiafen Gong, Zeynep Baskurt, Andriy Derkach, Angelina Pesevski and Lisa Strug

October, 2016

The Robust Variance Score (RVS) test is designed for association analysis for next generation sequencing (NGS) data when integrating sequence data from different studies. Compared to the usual score test, it uses a score test with a robust variance calculated from the expected genotype probability given the observed sequencing data $E(G_{ij}|D_{ij})$, rather than the genotype hard call. For detailed information about the RVS method, please see (Derkach et al. 2014).

The current version of the package is applicable to case-control studies only. To use the package, first you need to install it from github. You need to install package 'devtools' first in the R console and then you can use it to install the package by typing the command "install_github('Struglab/RVS')". The 'RVS' R package consists of three modules: (1) common and rare variant association analysis using RVS, (2) conventional association using variant calls, and (3) a simulation module. Module 1 takes a multi-sample variant call format file (VCF) created from all samples to be analyzed as input, obtains the expected genotype probabilities given the observed sequence data $E(G_{ij}|D_{ij})$ for the reported variants, and uses them in the RVS statistic for association analysis. Module 2 uses a conventional score test for association with variant calls rather than probabilities. For comparison; this module uses the plink (Purcell et al. 2007) format file as input. Module 3 simulates sequence data for different read depths and error rates. The main functions include: (1) function to process the vcf file (*vcf_process*); (2) association tests using the expected genotype probabilities which are the output of *vcf_process* (*RVS_asy*, *RVS_btrap* for common variants and *RVS_rare* for rare variants); (3) conventional association Rao score test using genotype hard calls (*regScore_Rao*, *regScore_btrap* for common variants and *regScore_rare* for rare variants), and (4) simulating sequence data (*generate_seqdata_OR*). The detailed usage of these functions and initialization of their argments are listed below.

Function to process the vcf file: vcf_process

This function uses a VCF file (eg. the output of the GATK (see DePristo et al. 2011) variant call pipeline) as input to extract the genotype likelihood $P(D_{ij}|G_{ij})$, ie. the probability of the observed data D_{ij} given genotype G_{ij} for sample i at variant j first, and then calculate the conditional expected genotype probability $E(G_{ij}|D_{ij})$ and genotype frequency $P(G_{ij})$ so that they can be used in the RVS test. To call the function, one needs to specify a VCF file (vcf_file), a file includes all the case IDs (caseID_file, each ID on one row), minor allele frequency cutoff (maf_cut) to distinguish common or rare variant, the missing rate used to filter out the variants (missing_th) and other information obtained from vcf file to call the function. Figure 1 is an example of a typical VCF file. A simple description of the VCF file and how to determine other parameters needed from the VCF file to call the *vcf_process* function is explained below.

The first parameter describing the VCF file is the number of header lines which start with # (nhead, nhead=19 in Figure 1). The last header line in the VCF file specifies the contents of each column in the remaining rows in the VCF file. Take Figure 1 for example, we can see from the nineteenth row that the first two columns are the chromosome and position of the variant (CHROM and POS), followed by identifier of the variant (ID, rs# if it is available), reference bases (REF), alternative calls (ALT), quality score (QUAL). The seventh column (named as 'FILTER') indicates the variants' filter status, where all the conditions used to filter a variant are listed in the header of VCF file (see rows 2 to 5 of the example VCF file). These conditions are specified

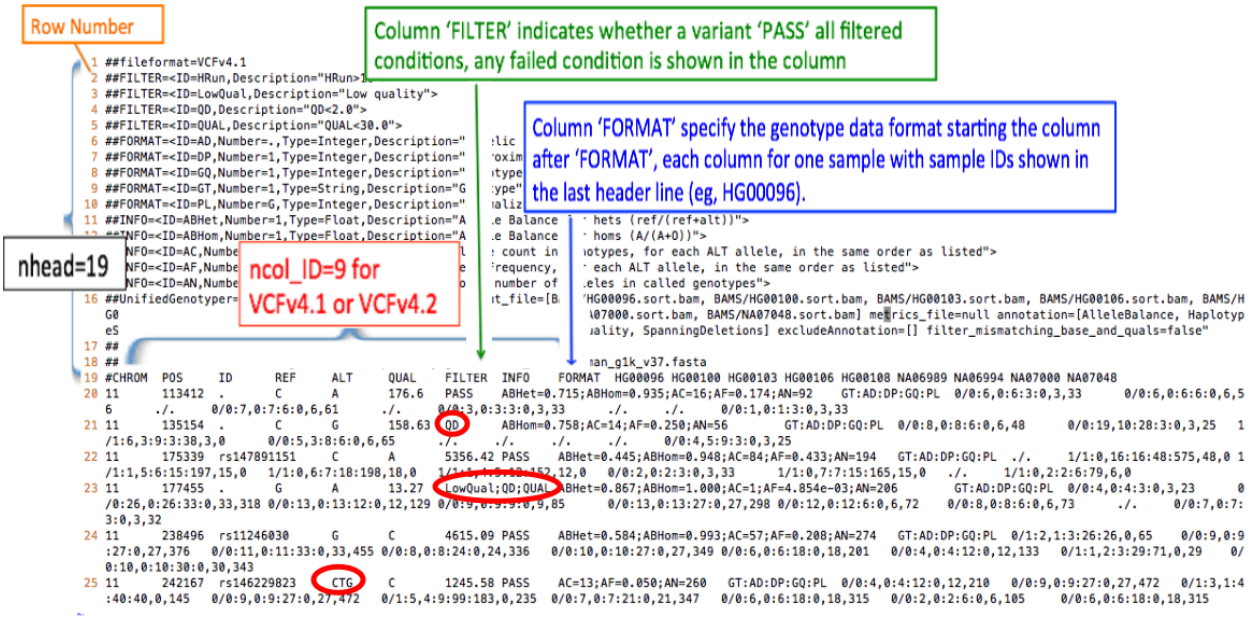

Figure 1: A VCF example

when making the variant call and can be obtained from the bioinformatician who processed the sequencing data and generated the VCF file(s). All variants with value 'PASS' in this column pass all the conditions (eg. the variant at chr11:113412), while variants that failed any condition will have that condition shown in this column (eg. variant at chr11:135154 failed the condition QD). The 'INFO' column includes additional keys that are calculated for each variant and each key's meaning can be found in the header part of VCF file as shown in row 11 -15 in the example. Each key is seperated by semi-colons with format 'key=value'. The column 'FORMAT' specifies the format of the genotype data in the rest of the columns of the vcf file. Each variant in the example here has a format of "GT:AD:DP:GQ:PL". Please note that this depends on the variant call pipeline. It is possible that different variants have different formats in the vcf file. After column 'FORMAT', each column is the data for each individual with their sample IDs listed in the last header row. The caseID_file mentioned earlier is a file that consists of all the IDs for case, one ID per row.

Parameter 'ncol_ID' is the number of columns before the genotype data for each individual (i.e., from column 'CHROM' to the column 'FORMAT', ncol_ID=9 in Figure 1); parameter 'nvar_tot' is the total number of variants in the VCF file (nvar_tot=6 in Figure 1). If their are too many variants in the vcf file, nvar_tot can also be a smaller number standing for the total number of variants you would like to work on; the number of variants you would like R to read in each time is denoted as nread (<=nvar_tot, if nread=nvar_tot, all variants will be read together). Besides these parameters, one needs another boolean indicator 'common' to indicate whether you would like to extract common or rare variants: common=TRUE for common variants and common=FALSE for rare variants. Once these paramaters are set, you can call the function *vcf_process*. This function will remove any variant that (1) fails any filters, that is, any variant without 'PASS' in the column 'FILTER' of the VCF file will be removed (e.g. variants at positions 135154 and 177455 in Figure 1); (2) has more than one alternative variant in column 'ALT'; (3) are biallelic short indels (e.g., variant at position 242167); and (4) has high missing rate (>missing_th). The remaining variants will be further filtered out if the standard deviation of the whole sample (including case and controls) are 0 and then return common or rare variants determined by the given minor allele frequency cut off (maf_cut) and common indicator.

Below is an example of variables needed to call the *vcf_process* function. Please change them according to your VCF file.

```
### initializing the VCF file information, arguments for vcf_process #######
vcf_file='~/Desktop/R_packages/RVS/example/example_1000snps.vcf'  ## the vcf file
```

```
caseID_file='~/Desktop/R_packages/RVS/example/caseID.txt'    ## the case IDs
nhead=128  ## the number of header lines in your VCF files (those starting with #)
ncol_ID=9     ## the number of columns before the data for each individuals.
missing_th=0.5## missing rate <missing_th in both case and controls will be kept
nvar_tot=1000 ## the total number of variants or the number of variants you want to work on
nread=300    ## number of variants R will read each time.
maf_cut=0.05     ## variants with maf>maf_cut are common variants, otherwise rare variants.
common=TRUE ## Indicating whether to take common variants (TRUE) or rare ones (FALSE).
```

After you call the function *vcf_process*, you should have some information shown on your screen to notify you on the progress of the program (please see the comments below starting with two # after calling the function). The vcf file (example_1000snps.vcf) and caseID file (caseID.txt) used here are also provided together with the package. You can download them and run the function as above to test whether your package is installed properly.

```
library('RVS')
```

```
## Loading required package: CompQuadForm

## Loading required package: MASS
```

```
var_com=vcf_process(vcf_file,caseID_file,nhead,ncol_ID, missing_th,
+                   nvar_tot, nread, maf_cut,common=TRUE)
```

```
## This VCF includes  169  samples with  56 cases.
## cases location index in all samples:
## 28 29 30 31 32 33 34 35 36 37 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 143
##
## while loop: 1 ,  300  variants read.
## Num.of.variant_kept_filtby_vcf_col2347= 155
## n_missing_keep= 1
## total dimension of case so far:  56 1
##
## while loop: 2 ,  600  variants read.
## Num.of.variant_kept_filtby_vcf_col2347= 123
## n_missing_keep= 18
## total dimension of case so far:  56 19
##
## while loop: 3 ,  900  variants read.
## Num.of.variant_kept_filtby_vcf_col2347= 98
## n_missing_keep= 53
## total dimension of case so far:  56 72
##
## while loop: 4 ,  1000  variants read.
## Num.of.variant_kept_filtby_vcf_col2347= 27
## n_missing_keep= 14
## total dimension of case so far:  56 86
##
## 86 SNPs out of  1000 SNPs are kept.
##
## 1 variant(s) is(are) removed because of homozygous call in all samples (ie. monomorphic).
## Expected genotype probabilities are calculated for 85  variants
## 60 out of 85 variants satisfies the MAF condition provided.
## There are  60 common variants.
```

Robust Variance Score (RVS) Association Test

There are a few association test functions included in the package: *RVS_asy* and *RVS_btrap* for the common variants and *RVS_rare* for rare varaints. They all use the conditional expected genotype probability, $E(G_{ij}|D_{ij})$, calculated from the VCF file for the association instead of the genotype hard calls. Each function concatenates two methods according to the option 'RVS'. They use the RVS method when RVS = 'TRUE', and the likelihood method as descriped in Skotte et al (Skotte, Korneliussen, and Albrechtsen 2012) in Genetic Epidemiology when RVS = 'FALSE'.

Association for Common Variants

The difference between *RVS_asy* and *RVS_btrap* are: *RVS_asy* uses the asymptotic distribution for the score test statistic to calculate the p-value while *RVS_btrap* uses the bootstrap method to calculate the p-values.

To use these functions, the arguments for each function include: phenotype vector (Y), the conditional expected genotype vector (Geno), population frequency (P), number of variants (nsnp) and RVS indicator (RVS, with value TRUE or FALSE). Function *RVS_btrap* need an extra variable: the number of bootstrap (nboot). The phenotype vector (Y), conditional expected genotype vector (Geno) and population frequency (P) are all outputs from calling function *vcf process*. Here I use the output of the function *vcf_process* called earlier to show how to call the function *RVS_btrap*. The function *RVS_asy* can be called similarly without the argument nboot.

```
## Replace RVS_btrap by RVS_asy if asymptotic method is preferred.
nboot=10000
Y=var_com[["Y"]];Geno=var_com[["Geno"]];P=var_com[["P"]];nsnp=var_com[["nsnp"]]
p.likely.btrap=rep(NA,nsnp)
p.RVS.btrap=rep(NA,nsnp)
for(i in 1:nsnp){
p.RVS.btrap[i]=RVS_btrap(Y,Geno[,i],P[i,],nboot,RVS='TRUE')
p.likely.btrap[i]<-RVS_btrap(Y,Geno[,i],P[i,],nboot,RVS='FALSE')
}

#### for plot observed vs expected -log(p)
exp.p=1:nsnp/(nsnp+1)
p.RVS.btrap1=p.RVS.btrap[order(p.RVS.btrap)]
p.likely.btrap1=p.likely.btrap[order(p.likely.btrap)]
plot(-log10(exp.p),-log10(p.likely.btrap1),xlab="-log10(expected pvalue)",
 ylab="-log10(observed pvalue)")
lines(-log10(exp.p),-log10(p.RVS.btrap1),col='red')
legend('topleft',legend=c('RVS','likelihood'),col=c('red','black'),pch=c('-','o'))
abline(a=0,b=1)
```

If multiple core computing resources are available, you can also use the following code for parallel computing to increase speed.

```
library('doMC')
library('foreach')
```

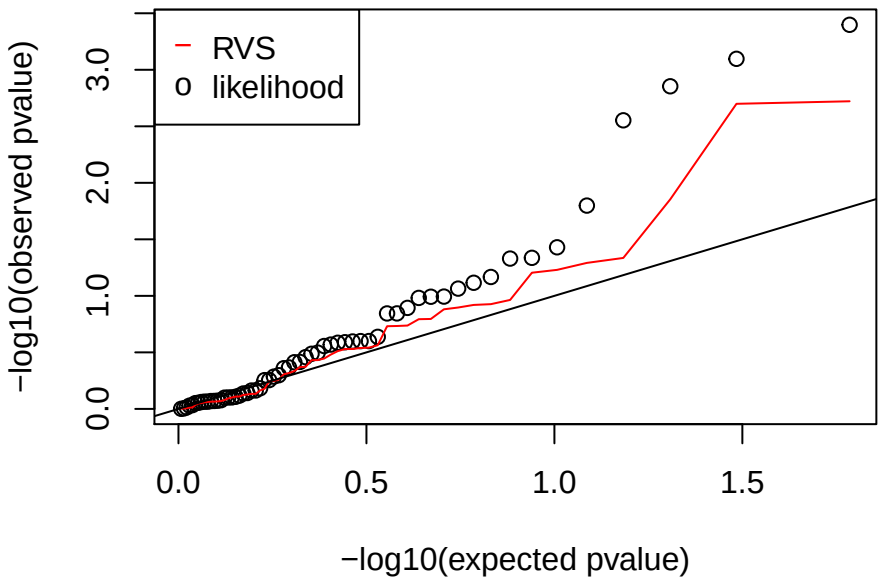

Figure 2: -log10(pvalue): observed vs expected for common variants

```
registerDoMC(25)  ## number of cores you would like to use

RVS_p_btrap2 = foreach(i=1:nsnp, .combine=rbind) %dopar% {
RVS_btrap(Y,Geno[,i],P[i,],nboot,RVS='TRUE')
}

likely_p_btrap2= foreach(i=1:nsnp, .combine=rbind) %dopar% {
RVS_btrap(Y,Geno[,i],P[i,],nboot,RVS='FALSE')
}
```

Association for Rare Variants

Association for rare variants is done by combining several rare variants together, so one more parameter needs to be specified: the number of variants one would like to group together for testing (variable njoint). In RVS, we group the njoint number of neighboring variants for rare variant association. If the total number of variants is indivisible by njoint, the remaining number of variants will be grouped together in the last group to form a larger group. In addition, it is possible that one group (for example, control) has homozygous calls only but the other group has alternative alleles for rare variants. When this happens, the matrix correlation will be infinity and to remove this homozygosity, we have two more arguments for *RVS_rare*: hom and multiplier. Both variables (hom and muliplier) have value 1 or 2 with default value of 1. We will manually change one individual's conditional expected genotype probability (hom=1 for the first non-NA individual and hom=2 for randomly-picked individuals with non-NA genotype) by dividing by 2 (multiplier=1) or multiplying by 2 (multiplier=2). Our similuations show that either combination of the choices work well by removing the homozygosity while keeping the association result more or less the same.

```
######### call vcf_process for rare variants and association by RVS test ###
rare=vcf_process(vcf_file,caseID_file,nhead, ncol_ID, missing_th,
+                nvar_tot, nread, maf_cut,common=FALSE)
# The message displayed on the screen from calling rare variants is
# simlar as the one for common variants, we omit it here.
```

```
Y=rare[["Y"]];Geno=rare[["Geno"]];P=rare[["P"]];nsnp=rare[["nsnp"]]
nboot=10000
p.RVS.rare=RVS_rare(Y,Geno,P,njoint=5,nboot,hom=1,multiplier=1)
```

Conventional Rao Score Test for genotype calls

To compare with the results of RVS association, functions using the genotype calls are also provided for your convenience, and they take Plink format file as input. Functions *regScore_Rao*, *regScore_perm* are provided for the common variants and *regScore_rare* for rare variants. Together with the package, an additive coding genotype hard call (example_additive_nofail.raw) is also provided. It is generated from the provided vcf file (example_1000snps.vcf) by vcftools to only include the variants with 'PASS' value in the 'FILTER' column. Function geno_process is provided to further filter variants that have duplication, homozygous call in all samples and high missing rate (>missing_cut). To call the function geno_process, one needs to provide 2 arguments: one is the file which lists the case IDs (one row for one sample without column names) and the other is genotype file with an additive coding (eg. file with extenstion raw generated by plink[1]).

In the following code, we first called function *geno_process* to obtain common variants, then use functions *regScore_Rao* and *regScore_perm* to do association test for the remaining variants and plot the p-values afterward. We also called *geno_process* with option 'common=F' for the rare variants and function *regScore_rare* for the association test.

```
#read in the hard call and filter variants
geno.file='~/Desktop/R_packages/RVS/example/example_additive_nofail.raw'
case_ID='~/Desktop/R_packages/RVS/example/caseID.txt'
geno1=geno_process(geno.file,case_ID,missing_cut=0.5,maf_cut=0.05,common=T)
```

```
## There are 466 variants in the inputed genotype files.
## 2 duplciate location.
## 58 indels to be removed.
## 16 monomorphic variants removed.
## 303 remaining variants are removed because missing rate higher than 0.5
## 60 remaining variants have MAF greater than 0.05
```

```
Y=geno1[["Y"]];geno=geno1[["geno"]];  nsnp1<-ncol(geno)

## association test
p.geno.asy=NULL
p.geno.btrap=NULL
```

[1] If only ped file is available, we also provide a function to convert the ped file to additive coding phenotype file (*ped_2_additive*).

```
nperm=10000
for(i in 1:nsnp1){
  p.geno.asy=c(p.geno.asy,regScore_Rao(Y,geno[,i]))
  p.geno.btrap=c(p.geno.btrap,regScore_perm(Y,geno[,i],nperm))
}

#### for plot observed vs expected -log(p)
exp.p=1:nsnp1/(nsnp1+1)
p.geno.btrap=p.geno.btrap[order(p.geno.btrap)]
plot(-log10(exp.p),-log10(p.geno.btrap),xlab='-log10(expected pvalue)',
     ylab='-log10(observed pvalue)',ylim=c(0,3))
lines(-log10(exp.p),-log10(p.RVS.btrap1),col='red')
legend('topleft',legend=c('RVS - expected probability','Score test - hard call'),
  col=c('red','black'),pch=c('-','o'))
abline(a=0,b=1)
```

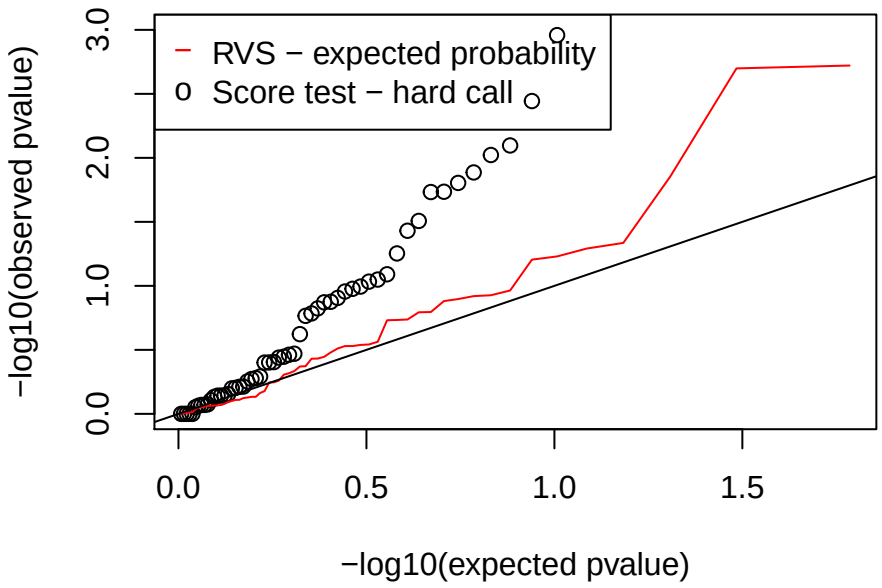

Figure 3: -log10(pvalue): observed vs expected from RVS and hard call

```
### rare variant
geno1=geno_process(geno.file,case_ID,missing_cut=0.5,maf_cut=0.05,common=F)
```

```
## There are 466 variants in the inputed genotype files.
## 2 duplciate location.
## 58 indels to be removed.
## 16 monomorphic variants removed.
## 303 remaining variants are removed because missing rate higher than 0.5
## 25 remaining variants have MAF less than 0.05
```

```
Y=geno1[["Y"]];geno2=geno1[["geno"]];  nsnp1<-ncol(geno1)
p.geno.rare=regScore_rare(Y,geno2,njoint=5,nperm=10000)
```

Some studies may consist of two datasets. This can happen when samples are sequenced at different times, by different technologies, etc. If you have samples from two datasets, you need to provide datasets in plink format (ie., sample1.ped/sample1.map and sample2.ped/sample2.map). For those who know how to use software plink, it is better to combine both datasets and exclude any samples that are not included in the study by plink first. For those who do not know plink, we also have a function *combine_twogeno* to combine both datasets and exclude any unwanted sample. You need to prepare one or two extra file(s) which consist(s) of the sample IDs that you would like to use(sample$i$_keep.txt i = 1 and/or 2), and the corresponding argument keepi=1, i = 1 and/or 2 when calling the function. If all the individuals in both datasets are included in the association study, one does not need to specify the arguments keepi, i=1, 2 and they can use the default value NA.

```
## the file names w/o extension
file1='~/Desktop/R_packages/RVS/example/sample1'
file2='~/Desktop/R_packages/RVS/example/sample2'
# combine two files together
# if only subset of sample1 needed for further anlysis,
# set keep1=1 and sample1_keep.txt should exist
# similar for sample2.
combined=combine_twogeno(file1,file2,missing_cut=0.5)
```

```
## snp 4  is missing in the first sample; removed.
## snp 48  is missing in the first sample; removed.
## snp 69  is missing in the first sample; removed.
##
## There are 33 common SNPs and  21  rare SNPs.
## All data are saved in list final, written to  data genotype_data.RData.
```

```
# association test, here only want to calculate p-values for at most 50 variants,
# because it is slow for too many variants in  R Markdown.
nsnp.geno=min(50,ncol(combined$common))
p_score_perm=rep(NA,nsnp.geno)
p_score_asy=rep(NA,nsnp.geno)
nperm=10000
Y=combined[['Y']]
common=combined[['common']]
for(i in 1:nsnp.geno){
 p_score_asy[i]=regScore_Rao(Y,common[,i])
 p_score_perm[i]=regScore_perm(Y,common[,i],nperm)
}
```

```
## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
```

```
plot(p_score_asy,p_score_perm)
abline(a=0,b=1)
```

Note, the warning message 'glm.fit:fitted probabilities numerically 0 or 1 occured' happened because of the perfect seperation of Y and X.

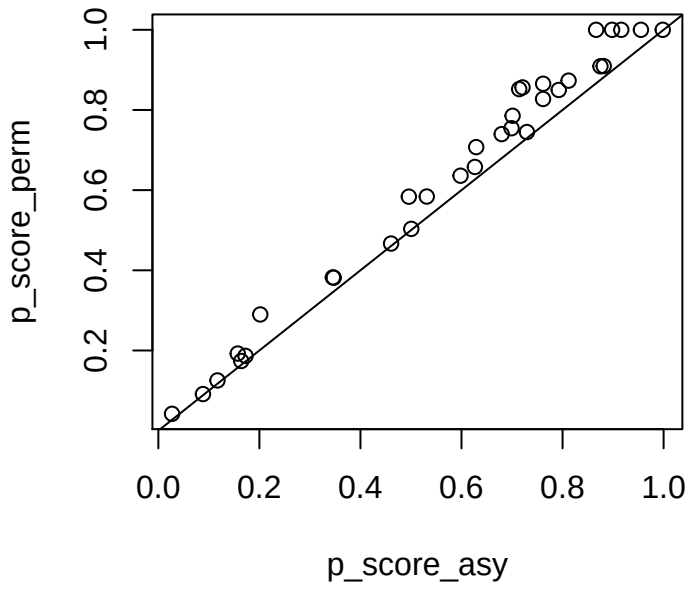

Figure 4: Comparison of two p values

Simulation sequencing data

Function provided for simulating sequencing data is *generate_seqdate_OR1*. You need to specify the population size (N), prevalence rate of the disease (preval), minor allele frequences for controls (mmafo) and odds ratio (OR) between phenotype and genotype to simulate the population, and the numbers of cases (ncase) and controls (ncont) that will be selected from the popluation. After the sample is chosen, the following variables are needed for simulating sequencing data: the mean and standard deviation of read depths for cases and controls (mdcase,sdcase; mdcont,sdcont), and the mean and standard deviation of the error rate of sequencing (me, sde). Each running of the function *generate_seqdate_OR1* can generate sequencing data for one snp. You can use for loop to sequence a larger number of snps (say nsnp), or use the parallel computing to save time as shown below. The function *generate_seqdata_OR1* returns a list including all the data for one variant: true genotype (geno), phenotype (pheno), read depth for each individual(rdepth_vector), expected genotype probability (exp_geno), population frequency (pop_freq). We also have a function *generate_seqdata_OR* for simulating nsnp number of SNPs for your convenience, and each row stands for data for one snp.

```
set.seed(123)
N=1000 #Integer. The number of population
preval=0.2 #A decimal between [0,1], prevalence rate of the disease.
nsnp=10   #Integer. The number of variants or bases.
```

```
ncase=50  #Integer. The number of individuals in sample 1 (called cases)
ncont=100 #Integer. The number of individuals in sample 2 (called controls)
mdcase=10 #Integer. The mean number of read depth for sample 1
sdcase=2 # Integer. The standard deviation of the read depth for sample 1
mdcont=50 #Integer. The mean number of read depth for sample 2
sdcont=3 # Integer. The standard deviation of the read depth for sample 2
me=runif(1,min=0,max=0.05)  #The mean error rate of sequencing, a decimal lies in [min,max] as specifie
sde=runif(1,min=0,max=0.05) #The standard deviation for the error rate, a decimal lies in [min,max] as

mmafco=runif(nsnp) # a vector of double decimal. The MAF for each variant in controls.
mmafco[mmafco>0.5]=1-mmafco[mmafco>0.5] # make sure MAF values fall in [0,0.5]

OR=1.5  ### Odds ratio in logit(Y)=bet0+bet*X, bet=log(OR)
Ntotal=ncase+ncont
```

```
library('doMC')
```

```
## Loading required package: foreach

## Loading required package: iterators

## Loading required package: parallel
```

```
library('foreach')
registerDoMC(40)

seqdata.null=generate_seqdata_OR(N,preval,ncase,ncont,mmafco,OR=1,
+                            mdcase,sdcase,mdcont,sdcont,me,sde,nsnp)

seqdata.alt=foreach(i=1:nsnp,.combine=rbindlist)%dopar%{
tmp<-generate_seqdata_OR1(N,preval,ncase,ncont,mmafco[i],OR,mdcase,sdcase,mdcont,sdcont,me,sde)
return(tmp)
}
```

Below is an example of how to use the simulated data for association test.

```
nboot=10000; p.null.rvs=rep(0,nsnp); p.alt.rvs=p.null.rvs
p.null.geno=rep(0,nsnp); p.alt.geno=p.null.rvs

Y=seqdata.null$pheno;  P=seqdata.null$pop_freq;nsnp=nrow(P);
Geno=seqdata.null$exp_geno; true1=seqdata.null$geno;
for(i in 1:nsnp)
{
  p.null.rvs[i]=RVS_btrap(Y[i,],Geno[i,],P[i,],nboot,RVS='TRUE')
  p.null.geno[i]=regScore_perm(Y[i,],true1[i,],nperm=nboot)
}

Y=seqdata.alt$pheno; P=seqdata.alt$pop_freq;nsnp=nrow(P)
Geno=seqdata.alt$exp_geno; true2=seqdata.alt$geno
for(i in 1:nsnp){
  p.alt.rvs[i]=RVS_btrap(Y[i,],Geno[i,],P[i,],nboot,RVS='TRUE')
  p.alt.geno[i]=regScore_perm(Y[i,],true2[i,],nperm=nboot)
}
```

```
## Warning: MAF of the SNP is 0.04999277 , try to group with other rare variants and use RVS_rare1.
```

```
t1=cbind(mmafco,p.null.geno,p.null.rvs,p.alt.geno,p.alt.rvs)
t1
```

```
##           mmafco p.null.geno p.null.rvs p.alt.geno  p.alt.rvs
##  [1,] 0.40897692  0.21567843 0.15588441 0.04489551 0.04249575
##  [2,] 0.11698260  0.20077992 0.16768323 0.44095590 0.41995800
##  [3,] 0.05953272  0.72842716 0.72902710 0.31196880 0.25307469
##  [4,] 0.04555650  0.06879312 0.06949305 0.38706129 0.26707329
##  [5,] 0.47189451  0.80621938 0.78122188 0.02019798 0.02239776
##  [6,] 0.10758096  0.14148585 0.07799220 0.04929507 0.03629637
##  [7,] 0.44856499  0.56024398 0.55484452 0.02709729 0.04379562
##  [8,] 0.45661474  0.02969703 0.04739526 0.01789821 0.02229777
##  [9,] 0.04316665  1.00000000 0.91560844 0.25657434 0.25217478
## [10,] 0.45333416  0.78402160 0.63283672 0.01179882 0.00969903
```

References

DePristo, M.A., E. Banks, R. Poplin, K.V. Garimella, J.R. Maguire, C. Hartl, A.A. Philippakis, et al. 2011. "A Framework for Variation Discovery and Genotyping Using Next-Generation DNA Sequencing Data." *Nat Genet* 43: 491–98.

Derkach, A., T. Chiang, J. Gong, L. Addis, S. Dobbins, I. Tomlinson, R. Houlston, D.K. Pal, and L.J. Strug. 2014. "Association Analysis Using Next-Generation Sequence Data from Publicly Available Control Groups: The Robust Variance Score Statistic." *Bioinformatics* 30: 2179–88.

Purcell, S., B. Neale, K. Todd-Brown, L. Thomas, M.A.R. Ferreira, D. Bender, J. Maller, et al. 2007. "PLINK: A Toolset for Whole-Genome Association and Population-Based Linkage Analysis." *American Journal of Human Genetics*, no. 3: 559–75.

Skotte, L., T.S. Korneliussen, and A. Albrechtsen. 2012. "Association Testing for Next-Generation Sequencing Data Using Score Statistics." *Genet Epidemiol,* 36: 430–37.