Package 'RVS'

Andriy Derkach, Ted Chiang, Jiafen Gong and Lisa Strug

January 15, 2016

Robust variance score (RVS) test is designed for association test of next generation sequencing (NGS) data when external control data is used. Compared to the regular score test, it uses score test with robust variance calculated from conditional genotype probability retrieved from the VCF file of the sequencing data, rather than the genotype hard call. Fore detailed information about the RVS method, please see the paper published in Bioinformatics in 2014: Association analysis using next-generation sequence data from publicly available control groups: the robust variance score statistic.

The current version of the package is applicable to case-control study only. To use the package, first you need to download the RVS package and install it by using command "install.packages('path/RVS0.0_tar.gz')" which 'path' is where you save the file RVS0.0_tar.gz. The package includes four modules: (1) function to process the vcf file (vcf_process); (2) association tests for conditional genotype probabilities which are the returns of vcf_process (RVS_asy, RVS_btrap for common variants and RVS_rare for rare variants); (3) association test (Rao Score test) for genotype hard call (regScore_Rao, regScore_btrap for common variants and regScore_rare for rare variants), and (4) simulating the sequence data (generate_seqdata_null and generate_seqdata_alt). The detailed usage of these functions and intitialization of the argments are listed as follows:

Function to process the vcf file: vcf_process

This module uses a VCF file (eg. the output of GATK variant call pipeline) as input to extract the genotype probability first and then calculate the conditional genotype probability so that they can be used in Robust Variance Score (RVS) test. One needs to provide the VCF file (vcf_file), a file to indicate all the case IDs (caseID_file), the number of header lines in the VCF file (IDline, those starting with #), the number of columns before the genotype data for each individual start (nhead), the missing rate in each subsample you want to use to filter out the SNPs (missing_th), total number of SNPs you would like to read (nsnp_tot), the number of SNPs you would like R to read in each time (nread <=nsnp, if nread=nsnp, all variants will be read together), minor allele frequency (MAF) cutoff for common or rare variant. Here is a example of how to determine the IDline and nhead from your VCF file.

Please note the last row starting with # in the VCF file (see Figure 1) specifies the contents of each column in the following rows. In this example, the first nine columns corresponding to chromosome (CHROM), position (POS), identifier of the variant (ID, rs# if it is available), reference base (REF), alternative base (ALT), quality score (QUAL), filtering status (FILTER), additional information with format key=data and each key is seperated by semi-colons (INFO, the keys shown here are determined by the options given to the variant calling algorithm when generating the VCF), and the format of rest columns (FORMAT). Columns after 'FORMAT' are genotype data for each indiviudal, with their sample IDs listed in this row. The caseID_file is a file with the list of case IDs (subset of HG****** and NA****** as shown in the above vcf file), one ID each row. Once these paramaters are set, you can use them to call the vcf_process. This function will remove any variant that (1) fails any filtlers, any variant without 'PASS' in column 'FILTER' of the VCF file will be removed (e.g., variants at positions 120799 and 196160 will be removed in Figure 1); (2) with more than one alternative variants in column 'ALT' (e.g., variant with position 120799); (3) biallelic short indels (e.g., variant at position 196163); and (4) with high missing rate (>missing_th). The ramained variants will

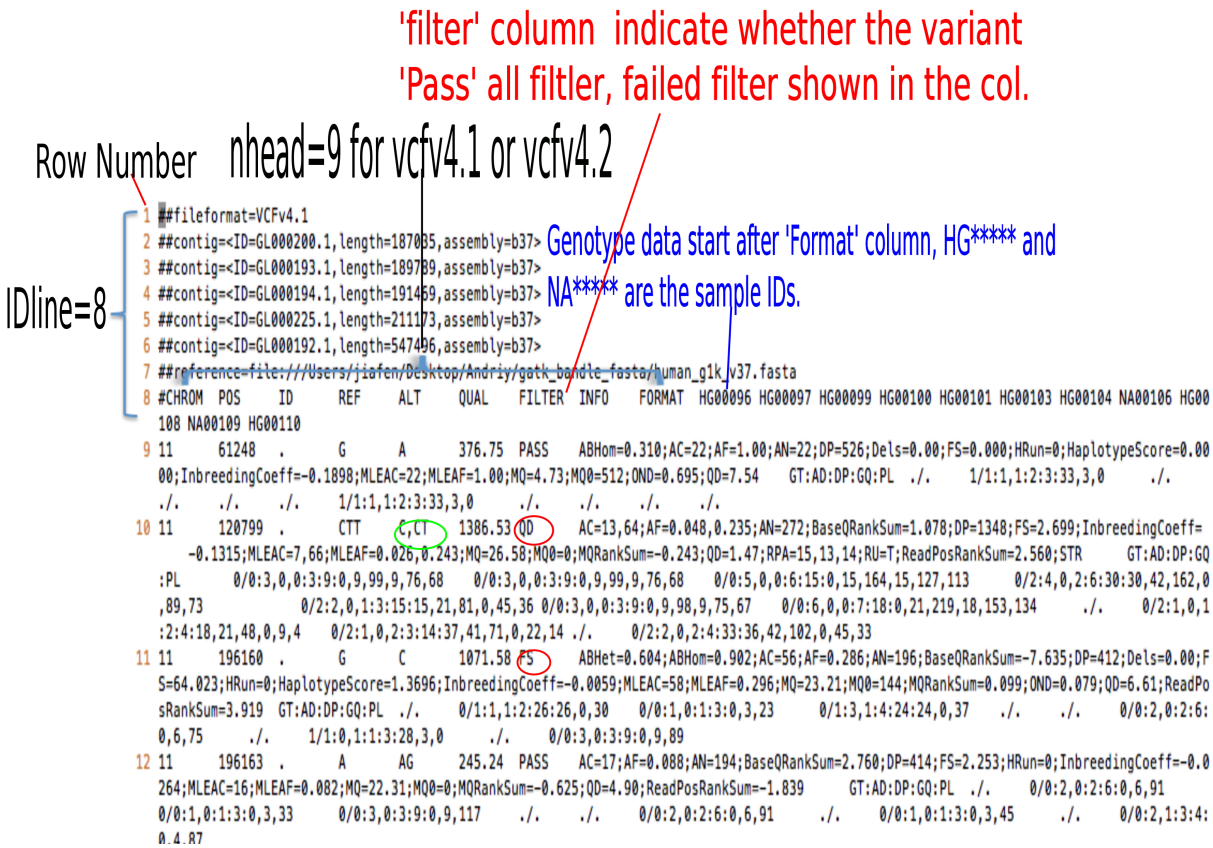

Figure 1: A simple vcf example

be further filtered out if the standard deviation of the whole sample (including case and controls) are 0 and then seperated into common and rare variants by the given MAF.

Please revise the values in the following box to address your dataset.

```
### initializing the VCF file information, arguments for vcf_process #######
vcf_file='~/Desktop/R_packages/RVS/example/example_1000snps.vcf'  ## the vcf file
caseID_file='/Users/jiafen/Desktop/R_packages/RVS/example/caseID.txt'    ## the case IDs
IDline=128  ## the number of header lines in your VCF files (those starting with #)
nhead=9     # the number of columns before the data for each individuals.
missing_th=0.5##filtering out the variants, missing rate <missing_th  will be kept
nsnp_tot=1000    ## the total number of variants or the number of variants you want to read
nread=300    ## number of variants R will read each time.
maf_cut=0.05     ##maf_cut used to the split the common and rare variants.
group='common' ## group used to indicate retrieve common variants or rare ones.
```

With the above parameters, you should have the ouput showed on your screen as displayed in the box after calling the function, two Rdata files named 'common_variants.Rdata' and 'rare_variants.RData' saved on your working directory, you can load them to test the robust variant test!

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

```
## Loading required package: CompQuadForm
```

```
## Loading required package: MASS
```

```
common=vcf_process(vcf_file,caseID_file,IDline, nhead, missing_th, nsnp_tot, nread, maf_cut,group='comm
```

```
## This VCF includes  169  samples with  56 cases!
## cases location 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
##
## loop= 1 ,vcf_col2347filt_keep= 155,n_missing_keep= 1
## while loop: 1  Until nows, totally read in 300  variants!
## total case dimension after filtering:  56 1
##
## loop= 2 ,vcf_col2347filt_keep= 123,n_missing_keep= 18
## while loop: 2  Until nows, totally read in 600  variants!
## total case dimension after filtering:  56 19
##
## loop= 3 ,vcf_col2347filt_keep= 98,n_missing_keep= 53
## while loop: 3  Until nows, totally read in 900  variants!
## total case dimension after filtering:  56 72
##
## loop= 4 ,vcf_col2347filt_keep= 27,n_missing_keep= 14
## while loop: 4  Until nows, totally read in 1000  variants!
## total case dimension after filtering:  56 86
##
## 86 SNPs out of  1000 SNPs are kept!
##
## There are  60 common variants!
##
## common variants info and expected probabilities are saved in file common_variants.RData!
```

```
#rare=vcf_process(vcf_file,caseID_file,IDline, nhead, missing_th, nsnp_tot, nread, maf_cut,group='rare'
```

The above vcf file (example_1000snps.vcf) and caseID file (caseID.txt) are also provided together with the package. To test whether your package is installed properly, you can download them and run the function as above. The two generated files are also provided in the data folder where your pacakge is installed, you can also load them to compare with those your just generated. To find out where your package is installed, use command `path1=.libPaths()`, the datasets generated on my end are save in the directory `path1/RVS/data/`.

Robust Variance Score (RVS) Association Test

There are a couple of functions for association test included in the package. For conditional genotype probability calculated from the VCF file, RVS_asy and RVS_btrap are available for the common variants and RVS_rare for rare varaints. Each function actually include two methods with different values for option 'RVS'. They use RVS method when RVS = 'TRUE', and likelihood method if RVS = 'FALSE' as descriped in Skotte's 2012 paper in Genetic Epidemiology.

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

The arguments for each function include: phenotype vector Y, genotype vector Geno, population freqency P, number of bootstrap nboot (not for RVS_asy) and method. The phenotype vector Y, genotype vector Geno and population frequency P are all outputs from vcf process. In the following I will load the RData saved by vcf_process for example to show how to use these functions.

```
######### load genotype data and initializing phenotype Y for RVS test ###
#path1=.libPaths()
#load(paste0(path1,'/RVS/data/Common_variants'))  ## load the dataset provided with RVS package.
load('common_variants.RData')  ## load the dataset generated in calling vcf_process.
```

Here I load the output of vcf_process in the previous part of the document. If you do not want to run vcf_process, use the commented commands can load the output of vcf_process which are provided together with the R package. For multiple SNPs, we could use a for loop to run the association for each SNP as shown below,

```
## Replace RVS_btrap by RVS_asy if asyptotic method wanted.
nboot=10000
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)
```

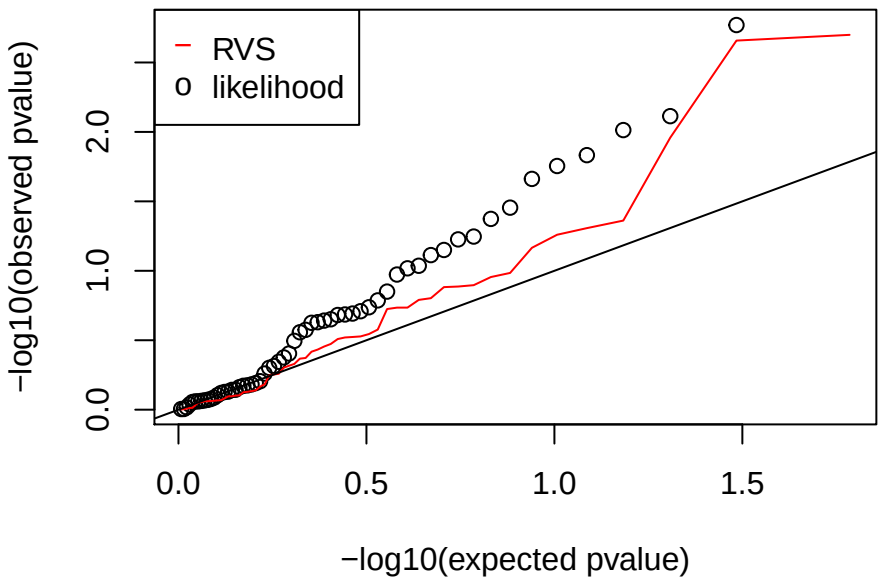

Figure 2: -log10(pvalue): observed vs expected

The RVS from asymptotic distribution are very similar as above, except replacing the function by RVS_asy and remove the argument nboot. If multiple core computing resource is available, you can also using the following code for the parallel computing.

```
library('doMC')
library('foreach')
registerDoMC(25)  ## number of cores you would like to use

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

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

Association for Rare Variants

We will combine several rare variants for the association test, so we need to specify one more parameter: number of snp you would like to group together for the testing (njoint). In RVS, we group the njoint number of neighboring variants for rare variant association. If the total number of SNP can not be divided by njoint, the remainder number of the SNPs will be grouped together to the last group to form a larger group.

```
######### load genotype data and initializing phenotype Y for RVS test ###
#path1=.libPaths()
#load(paste0(path1,'/RVS/data/Rare_variants'))  ## load the dataset provided with RVS package.
rare=vcf_process(vcf_file,caseID_file,IDline, nhead, missing_th, nsnp_tot, nread, maf_cut,group='rare')
load('rare_variants.RData')  ## load the dataset generated in calling vcf_process.

## njoint is the number of variants grouped for association
## if Num. of variants is not multiple of njoint, the remaining variants will be grouped to the last gr
p.RVS.rare=RVS_rare(Y,Geno,P,njoint=5,nboot)
```

Regular Score Test (Rao) for genotype hard calls

To compare with our RVS results, functions working on the genotype hard calls are also provided for your convenience. Functions regScore_Rao, regScore_perm are available 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 example_1000snps.vcf by vcftools to only include the variants with 'PASS' value in filter column. In function geno_process, it will add the phenotype into the datasets, filtering variants by missingness. In the following code, we call geno_process, then regScore_Rao and regScore_perm are used to do association for all variants.

```
#reorder the genotype data
geno.file='/Users/jiafen/Desktop/R_packages/RVS/example/example_additive_nofail.raw'
case_ID='/Users/jiafen/Desktop/R_packages/RVS/example/caseID.txt'
geno=geno_process(geno.file,case_ID,missing_cut=0.5,maf_cut=0.05,common=T)

## association test
p.geno.asy=NULL
p.geno.btrap=NULL
nperm=10000
nsnp1<-ncol(geno$geno)
for(i in 1:nsnp1){
  p.geno.asy=c(p.geno.asy,regScore_Rao(geno$Y,geno$geno[,i])$p_Rao)
  p.geno.btrap=c(p.geno.btrap,regScore_perm(geno$Y,geno$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)',y
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'),p
abline(a=0,b=1)
```

If you have samples from two datasets in plink format (ped/map) (this can happen when they are genotyping at different times, sequencing by different techniques etc) and would like to use their common variants to do the association, here is an example.

```
## the file names w/o extension
file1='/Users/jiafen/Desktop/R_packages/RVS/example/sample1'
file2='/Users/jiafen/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, then keep2=1 needed.
combined=combine_twogeno(file1,file2,keep1=1,missing_cut=0.5)
```

```
## snp 124  is monomorphic in the whole samples and removed!
## snp 289  is monomorphic in the whole samples and removed!
## snp 453  is monomorphic in the whole samples and removed!
## snp 457  is monomorphic in the whole samples and removed!
## snp 504  is biallelic in each sample but has more than 2 alleles in whole sample! removed!
## snp 544  is biallelic in each sample but has more than 2 alleles in whole sample! removed!
## snp 644  is biallelic in each sample but has more than 2 alleles in whole sample! removed!
## snp 682  is monomorphic in the whole samples and removed!
## snp 704  is monomorphic in the whole samples and removed!
## snp 708  is monomorphic in the whole samples and removed!
## snp 713  is monomorphic in the whole samples and removed!
## snp 718  is monomorphic in the whole samples and removed!
## snp 719  is monomorphic in the whole samples and removed!
## snp 723  is monomorphic in the whole samples and removed!
## snp 741  is biallelic in each sample but has more than 2 alleles in whole sample! removed!
## snp 742  is monomorphic in the whole samples and removed!
## snp 744  is monomorphic in the whole samples and removed!
## snp 746  is monomorphic in the whole samples and removed!
## snp 748  is monomorphic in the whole samples and removed!
## snp 757  is monomorphic in the whole samples and removed!
## snp 762  is monomorphic in the whole samples and removed!
## snp 764  is monomorphic in the whole samples and removed!
## snp 777  is monomorphic in the whole samples and removed!
## snp 804  is monomorphic in the whole samples and removed!
## snp 819  is monomorphic in the whole samples and removed!
## snp 820  is monomorphic in the whole samples and removed!
## snp 868  is monomorphic in the whole samples and removed!
## snp 874  is monomorphic in the whole samples and removed!
## snp 881  is monomorphic in the whole samples and removed!
## snp 892  is missing in the second sample and removed!
## snp 895  is missing in the second sample and removed!
## snp 899  is monomorphic in the whole samples and removed!
## snp 915  is monomorphic in the whole samples and removed!
## snp 926  is monomorphic in the whole samples and removed!
## snp 989  is biallelic in each sample but has more than 2 alleles in whole sample! removed!
## snp 1081  is monomorphic in the whole samples and removed!
##
## There are 640 common SNPs and  528  rare SNPs!
## All data are saved in binary data genotype_data, can be reached by final$common, final$rare and final
```

```
#association, only want to work on at most 50 variants to fast the process.
nsnp.geno=50 #min(50,ncol(combined$common))
p_score_perm=rep(NA,nsnp.geno)
p_score_asy=rep(NA,nsnp.geno)
nperm=1000
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)
}

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

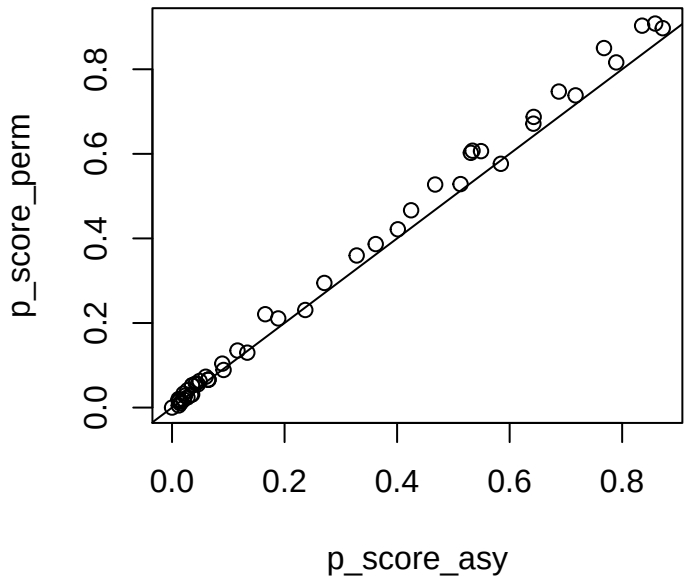

Figure 3: Comparison of two p values

Simulation sequencing data

There are two functions to simulation sequencing data: generate_seqdata_null and generate_seqdata_alt, where the first function assume two subsamples sequenced with the different read depths are from the same

population: that is, the minor allele frequency (MAF) for each SNP in the two subsamples are the same, but the second function assume two subsamples have different MAFs. The inputs for each function are shown as below.

```
set.seed(123)
nsnp=10   #Integer. Number of SNPs or bases.
ncase=50  #Integer. Number of individuals in sample 1 (called cases)
ncont=100 #Integer. 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)  #double decimal with value in [0,1]. The mean error rate of each snp.
sde=runif(1) # double decimal with value in [0,1]. The standard deviation for the error rate.
mmaf=runif(nsnp)  # a vector of double decimal. The MAF for each SNPs in null samples.
mmaf[mmaf>0.5]=1-mmaf[mmaf>0.5] # make sure MAF values fall in [0,0.5].

mmafa=runif(nsnp)  # a vector of double decimal. The MAF for each SNP in cases.
mmafa[mmafa>0.5]=1-mmafa[mmafa>0.5] # make sure MAF values fall in [0,0.5]
mmafo=runif(nsnp) # a vector of double decimal. The MAF for each SNP in controls.
mmafo[mmafo>0.5]=1-mmaf[mmafo>0.5] # make sure MAF values fall in [0,0.5]
```

With different arguments for MAF (generate_seqdata_null only need to provides one MAF as shown above as 'mmaf' and generate_seqdata_alt need two MAFs: mmafa and mmafo), each funtion returns a list including the expected genotype probabilitie for these SNPs (MM), the population frequency for each SNP (P) and the original genotype (G).

```
seqdata.null<-generate_seqdata_null(nsnp,ncase,ncont,mdcase,sdcase,mdcont,sdcont,me,sde,mmaf)
seqdata.alt<-generate_seqdata_alt(nsnp,ncase,ncont,mdcase,sdcase,mdcont,sdcont,me,sde,mmafa,mmafo)
names(seqdata.null)
```