A Tutorial for the R/Bioconductor Package SNPRelate

Xiuwen Zheng
Department of Biostatistics
University of Washington

Jan 13, 2015

Contents

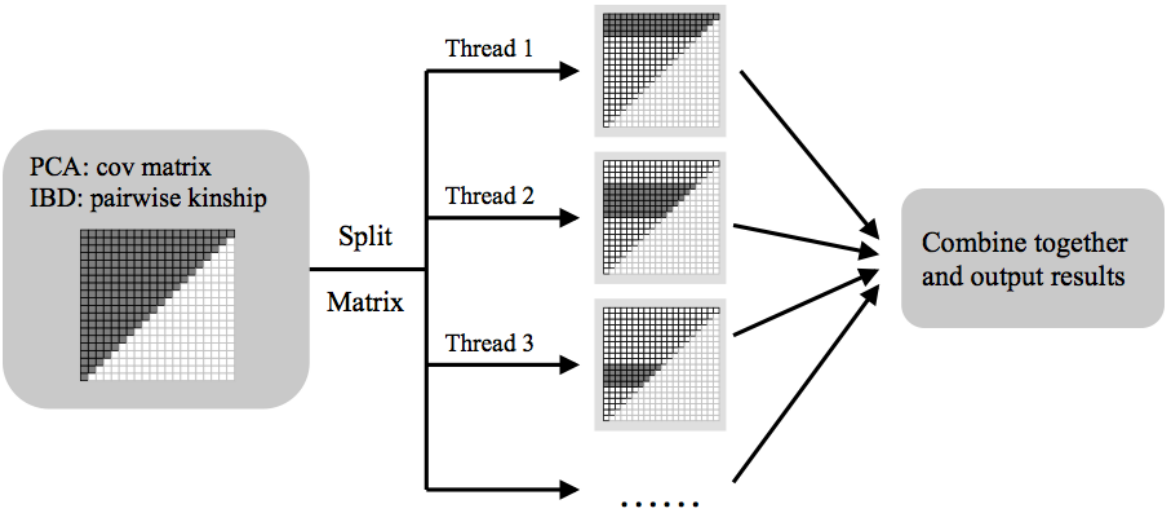


Figure 1: Flowchart of parallel computing for principal component analysis and identity-by-descent analysis.

1 Overview

Genome-wide association studies (GWAS) are widely used to help determine the genetic basis of diseases and traits, but they pose many computational challenges. We developed *gdsfmt* and *SNPRelate* (high-performance computing R packages for multi-core symmetric multiprocessing computer architectures) to accelerate two key computations in GWAS: principal component analysis (PCA) and relatedness analysis using identity-by-descent (IBD) measures[1]. The kernels of our algorithms are written in C/C++ and have been highly optimized. The calculations of the genetic covariance matrix in PCA and pairwise IBD coefficients are split into non-overlapping parts and assigned to multiple cores for performance acceleration, as shown in Figure 1. Benchmarks show the uniprocessor implementations of PCA and IBD are ∼8 to 50 times faster than the implementations provided in the popular *EIGENSTRAT* (v3.0) and *PLINK* (v1.07) programs respectively, and can be sped up to 30∼300 folds by utilizing multiple cores.

GDS is also used by an R/Bioconductor package *GWASTools* as one of its data storage formats[2,3]. *GWASTools* provides many functions for quality control and analysis of GWAS, including statistics by SNP or scan, batch quality, chromosome anomalies, association tests, etc.

R is the most popular statistical programming environment, but one not typically optimized for high performance or parallel computing which would ease the burden of large-scale GWAS calculations. To overcome these limitations we have developed a project named CoreArray (`http://corearray.sourceforge.net/`) that includes two R packages: *gdsfmt* to provide efficient, platform independent memory and file management for genome-wide numerical data, and *SNPRelate* to solve large-scale, numerically intensive GWAS calculations (i.e., PCA and IBD) on multi-core symmetric multiprocessing (SMP) computer architectures.

This vignette takes the user through the relatedness and principal component analysis used for genome

wide association data. The methods in these vignettes have been introduced in the paper of Zheng *et al.* (2012)[1]. For replication purposes the data used here are taken from the HapMap Phase II project. These data were kindly provided by the Center for Inherited Disease Research (CIDR) at Johns Hopkins University and the Broad Institute of MIT and Harvard University (Broad). The data supplied here should not be used for any purpose other than this tutorial.

2 Preparing Data

2.1 Data formats used in SNPRelate

To support efficient memory management for genome-wide numerical data, the *gdsfmt* package provides the genomic data structure (GDS) file format for array-oriented bioinformatic data, which is a container for storing annotation data and SNP genotypes. In this format each byte encodes up to four SNP genotypes thereby reducing file size and access time. The GDS format supports data blocking so that only the subset of data that is being processed needs to reside in memory. GDS formatted data is also designed for efficient random access to large data sets.

```
# Load the R packages: gdsfmt and SNPRelate
library(gdsfmt)
library(SNPRelate)

## SNPRelate -- supported by Streaming SIMD Extensions 2 (SSE2)
```

Here is a typical GDS file:

```
snpgdsSummary(snpgdsExampleFileName())

## The file name: /tmp/RtmpqtRuP6/Rinst14db3eaa810c/SNPRelate/extdata/hapmap_geno.gds
## The total number of samples: 279
## The total number of SNPs: 9088
## SNP genotypes are stored in SNP-major mode (Sample X SNP).
```

`snpgdsExampleFileName()` returns the file name of a GDS file used as an example in *SNPRelate*, and it is a subset of data from the HapMap project and the samples were genotyped by the Center for Inherited Disease Research (CIDR) at Johns Hopkins University and the Broad Institute of MIT and Harvard University (Broad). `snpgdsSummary()` summarizes the genotypes stored in the GDS file. "Individual-major mode" indicates listing all SNPs for an individual before listing the SNPs for the next individual, etc. Conversely, "SNP-major mode" indicates listing all individuals for the first SNP before listing all individuals for the second SNP, etc. Sometimes "SNP-major mode" is more computationally efficient than "individual-major model". For example, the calculation of genetic covariance matrix deals with genotypic data SNP by SNP, and then "SNP-major mode" should be more efficient.

```
# Open a GDS file
(genofile <- snpgdsOpen(snpgdsExampleFileName()))

## File: /tmp/RtmpqtRuP6/Rinst14db3eaa810c/SNPRelate/extdata/hapmap_geno.gds
```

```
## +    [  ] *
## |--+ sample.id   { FStr8 279 ZIP(23.10%) }
## |--+ snp.id   { Int32 9088 ZIP(34.76%) }
## |--+ snp.rs.id   { FStr8 9088 ZIP(42.66%) }
## |--+ snp.position   { Int32 9088 ZIP(94.73%) }
## |--+ snp.chromosome   { UInt8 9088 ZIP(0.94%) } *
## |--+ snp.allele   { FStr8 9088 ZIP(14.45%) }
## |--+ genotype   { Bit2 279x9088 } *
## |--+ sample.annot   [ data.frame ] *
## |  |--+ sample.id   { FStr8 279 ZIP(23.10%) }
## |  |--+ family.id   { FStr8 279 ZIP(28.37%) }
## |  |--+ father.id   { FStr8 279 ZIP(12.98%) }
## |  |--+ mother.id   { FStr8 279 ZIP(12.86%) }
## |  |--+ sex   { FStr8 279 ZIP(28.32%) }
## |  |--+ pop.group   { FStr8 279 ZIP(7.89%) }
```

The output lists all variables stored in the GDS file. At the first level, it stores variables `sample.id`, `snp.id`, etc. The additional information are displayed in the braces indicating data type, size, compressed or not + compression ratio. The second-level variables `sex` and `pop.group` are both stored in the folder of `sample.annot`. All of the functions in *SNPRelate* require a minimum set of variables in the annotation data. The minimum required variables are

- `sample.id`, a unique identifier for each sample.
- `snp.id`, a unique identifier for each SNP.
- `snp.position`, the base position of each SNP on the chromosome, and 0 for unknown position; it does not allow NA.
- `snp.chromosome`, an integer or character mapping for each chromosome. Integer: numeric values 1-26, mapped in order from 1-22, 23=X,24=XY (the pseudoautosomal region), 25=Y, 26=M (the mitochondrial probes), and 0 for probes with unknown positions; it does not allow NA. Character: "X", "XY", "Y" and "M" can be used here, and a blank string indicating unknown position.
- genotype, a SNP genotypic matrix. SNP-major mode: $n_{sample} \times n_{snp}$, individual-major mode: $n_{snp} \times n_{sample}$.

Users can define the numeric chromosome codes which are stored with the variable `snp.chromosome` as its attributes when `snp.chromosome` is numeric only. For example, `snp.chromosome` has the attributes of chromosome coding:

```
# Get the attributes of chromosome coding
get.attr.gdsn(index.gdsn(genofile, "snp.chromosome"))
```

```
## $autosome.start
## [1] 1
##
## $autosome.end
## [1] 22
```

```
##
## $X
## [1] 23
##
## $XY
## [1] 24
##
## $Y
## [1] 25
##
## $M
## [1] 26
##
## $MT
## [1] 26
```

`autosome.start` is the starting numeric code of autosomes, and `autosome.end` is the last numeric code of autosomes. `put.attr.gdsn` can be used to add a new attribute or modify an existing attribute.

There are four possible values stored in the variable `genotype`: 0, 1, 2 and 3. For bi-allelic SNP sites, "0" indicates two B alleles, "1" indicates one A allele and one B allele, "2" indicates two A alleles, and "3" is a missing genotype. For multi-allelic sites, it is a count of the reference allele (3 meaning no call). "Bit2" indicates that each byte encodes up to four SNP genotypes since one byte consists of eight bits.

```
# Take out genotype data for the first 3 samples and the first 5 SNPs
(g <- read.gdsn(index.gdsn(genofile, "genotype"), start=c(1,1), count=c(5,3)))
```

```
##      [,1] [,2] [,3]
## [1,]    2    1    0
## [2,]    1    1    0
## [3,]    2    1    1
## [4,]    2    1    1
## [5,]    0    0    0
```

Or take out genotype data with sample and SNP IDs, and four possible values are returned 0, 1, 2 and NA (3 is replaced by NA):

```
g <- snpgdsGetGeno(genofile, sample.id=..., snp.id=...)
```

```
# Get the attribute of genotype
get.attr.gdsn(index.gdsn(genofile, "genotype"))
```

```
## $sample.order
## NULL
```

The returned value could be either "snp.order" or "sample.order", indicating individual-major mode (snp is the first dimension) and SNP-major mode (sample is the first dimension) respectively.

```
# Take out snp.id
head(read.gdsn(index.gdsn(genofile, "snp.id")))
```

```
## [1] 1 2 3 4 5 6
```

```
# Take out snp.rs.id
head(read.gdsn(index.gdsn(genofile, "snp.rs.id")))
```

```
## [1] "rs1695824"  "rs13328662" "rs4654497"  "rs10915489" "rs12132314" "rs12042555"
```

There are two additional variables:

- `snp.rs.id`, a character string for reference SNP ID that may not be unique.
- `snp.allele`, it is not necessary for the analysis, but it is necessary when merging genotypes from different platforms. The format of `snp.allele` is "A allele/B allele", like "T/G" where T is A allele and G is B allele.

The information of sample annotation can be obtained by the same function `read.gdsn`. For example, population information. "FStr8" indicates a character-type variable.

```
# Read population information
pop <- read.gdsn(index.gdsn(genofile, path="sample.annot/pop.group"))
table(pop)
```

```
## pop
## CEU HCB JPT YRI
##  92  47  47  93
```

```
# Close the GDS file
snpgdsClose(genofile)
```

2.2 Create a GDS File of Your Own

2.2.1 snpgdsCreateGeno

The function `snpgdsCreateGeno` can be used to create a GDS file. The first argument should be a numeric matrix for SNP genotypes. There are possible values stored in the input genotype matrix: 0, 1, 2 and other values. "0" indicates two B alleles, "1" indicates one A allele and one B allele, "2" indicates two A alleles, and other values indicate a missing genotype. The SNP matrix can be either $n_{sample} \times n_{snp}$ (snpfirstdim=FALSE, the argument in `snpgdsCreateGeno`) or $n_{snp} \times n_{sample}$ (snpfirstdim=TRUE).

For example,

```
# Load data
data(hapmap_geno)

# Create a gds file
snpgdsCreateGeno("test.gds", genmat = hapmap_geno$genotype,
```

```
    sample.id = hapmap_geno$sample.id, snp.id = hapmap_geno$snp.id,
    snp.chromosome = hapmap_geno$snp.chromosome,
    snp.position = hapmap_geno$snp.position,
    snp.allele = hapmap_geno$snp.allele, snpfirstdim=TRUE)

# Open the GDS file
(genofile <- snpgdsOpen("test.gds"))
```

```
## File: /tmp/RtmpqtRuP6/Rbuild14db5253eb32/SNPRelate/vignettes/test.gds
## +    [  ] *
## |--+ sample.id   { VStr8 279 ZIP(29.89%) }
## |--+ snp.id   { VStr8 1000 ZIP(42.42%) }
## |--+ snp.position   { Float64 1000 ZIP(55.97%) }
## |--+ snp.chromosome   { Int32 1000 ZIP(2.00%) }
## |--+ snp.allele   { VStr8 1000 ZIP(13.85%) }
## |--+ genotype   { Bit2 1000x279 } *
```

```
# Close the GDS file
snpgdsClose(genofile)
```

2.2.2 Uses of the Functions in the Package gdsfmt

In the following code, the functions `createfn.gds`, `add.gdsn`, `put.attr.gdsn`, `write.gdsn` and `index.gdsn` are defined in the package *gdsfmt*:

```
# Create a new GDS file
newfile <- createfn.gds("your_gds_file.gds")

# add a flag
put.attr.gdsn(newfile$root, "FileFormat", "SNP_ARRAY")

# Add variables
add.gdsn(newfile, "sample.id", sample.id)
add.gdsn(newfile, "snp.id", snp.id)
add.gdsn(newfile, "snp.position", snp.position)
add.gdsn(newfile, "snp.chromosome", snp.chromosome)
add.gdsn(newfile, "snp.allele", c("A/G", "T/C", ...))


##########################################################################
# Create a snp-by-sample genotype matrix

# Add genotypes
var.geno <- add.gdsn(newfile, "genotype",
    valdim=c(length(snp.id), length(sample.id)), storage="bit2")
```

```
# Indicate the SNP matrix is snp-by-sample
put.attr.gdsn(var.geno, "snp.order")

# Write SNPs into the file sample by sample
for (i in 1:length(sample.id))
{
    g <- ...
    write.gdsn(var.geno, g, start=c(1,i), count=c(-1,1))
}


############################################################################
# OR, create a sample-by-snp genotype matrix

# Add genotypes
var.geno <- add.gdsn(newfile, "genotype",
    valdim=c(length(sample.id), length(snp.id)), storage="bit2")

# Indicate the SNP matrix is sample-by-snp
put.attr.gdsn(var.geno, "sample.order")

# Write SNPs into the file sample by sample
for (i in 1:length(snp.id))
{
    g <- ...
    write.gdsn(var.geno, g, start=c(1,i), count=c(-1,1))
}



# Get a description of chromosome codes
#   allowing to define a new chromosome code, e.g., snpgdsOption(Z=27)
option <- snpgdsOption()
var.chr <- index.gdsn(newfile, "snp.chromosome")
put.attr.gdsn(var.chr, "autosome.start", option$autosome.start)
put.attr.gdsn(var.chr, "autosome.end", option$autosome.end)
for (i in 1:length(option$chromosome.code))
{
    put.attr.gdsn(var.chr, names(option$chromosome.code)[i],
        option$chromosome.code[[i]])
}

# Add your sample annotation
samp.annot <- data.frame(sex = c("male", "male", "female", ...),
```

```
    pop.group = c("CEU", "CEU", "JPT", ...), ...)
add.gdsn(newfile, "sample.annot", samp.annot)

# Add your SNP annotation
snp.annot <- data.frame(pass=c(TRUE, TRUE, FALSE, FALSE, TRUE, ...), ...)
add.gdsn(newfile, "snp.annot", snp.annot)


# Close the GDS file
closefn.gds(newfile)
```

2.3 Format conversion from PLINK binary files

The *SNPRelate* package provides a function `snpgdsBED2GDS` for converting a PLINK binary file to a GDS file:

```
# The PLINK BED file, using the example in the SNPRelate package
bed.fn <- system.file("extdata", "plinkhapmap.bed", package="SNPRelate")
fam.fn <- system.file("extdata", "plinkhapmap.fam", package="SNPRelate")
bim.fn <- system.file("extdata", "plinkhapmap.bim", package="SNPRelate")
```

Or, uses your own PLINK files:

```
bed.fn <- "C:/your_folder/your_plink_file.bed"
fam.fn <- "C:/your_folder/your_plink_file.fam"
bim.fn <- "C:/your_folder/your_plink_file.bim"
```

```
# Convert
snpgdsBED2GDS(bed.fn, fam.fn, bim.fn, "test.gds")
```

```
## Start snpgdsBED2GDS ...
##  BED file: "/tmp/RtmpqtRuP6/Rinst14db3eaa810c/SNPRelate/extdata/plinkhapmap.bed" in the
##  FAM file: "/tmp/RtmpqtRuP6/Rinst14db3eaa810c/SNPRelate/extdata/plinkhapmap.fam", DONE.
##  BIM file: "/tmp/RtmpqtRuP6/Rinst14db3eaa810c/SNPRelate/extdata/plinkhapmap.bim", DONE.
## Mon Jan 19 21:04:52 2015  store sample id, snp id, position, and chromosome.
##  start writing: 279 samples, 5000 SNPs ...
##    Mon Jan 19 21:04:52 2015 0%
##    Mon Jan 19 21:04:52 2015 100%
## Mon Jan 19 21:04:52 2015  Done.
## Optimize the access efficiency ...
## Clean up the fragments of GDS file:
##  open the file "test.gds" (size: 380417).
##  # of fragments in total: 39.
##  save it to "test.gds.tmp".
##  rename "test.gds.tmp" (size: 380165).
```

```
##  # of fragments in total: 18.
```

```
# Summary
snpgdsSummary("test.gds")
```

```
## The file name: /tmp/RtmpqtRuP6/Rbuild14db5253eb32/SNPRelate/vignettes/test.gds
## The total number of samples: 279
## The total number of SNPs: 5000
## SNP genotypes are stored in individual-major mode (SNP X Sample).
```

2.4 Format conversion from VCF files

The *SNPRelate* package provides a function `snpgdsVCF2GDS` to reformat a VCF file. There are two options for extracting markers from a VCF file for downstream analyses: (1) to extract and store dosage of the reference allele only for biallelic SNPs and (2) to extract and store dosage of the reference allele for all variant sites, including bi-allelic SNPs, multi-allelic SNPs, indels and structural variants.

```
# The VCF file, using the example in the SNPRelate package
vcf.fn <- system.file("extdata", "sequence.vcf", package="SNPRelate")
```

Or, uses your own VCF file:

```
vcf.fn <- "C:/your_folder/your_vcf_file.vcf"
```

```
# Reformat
snpgdsVCF2GDS(vcf.fn, "test.gds", method="biallelic.only")
```

```
## VCF Format --> SNP GDS Format
## Method: exacting biallelic SNPs
## Number of samples: 3
## Parsing "/tmp/RtmpqtRuP6/Rinst14db3eaa810c/SNPRelate/extdata/sequence.vcf" ...
##  import 2 variants.
## + genotype   { Bit2 3x2 } *
## Optimize the access efficiency ...
## Clean up the fragments of GDS file:
##  open the file "test.gds" (size: 2471).
##  # of fragments in total: 39.
##  save it to "test.gds.tmp".
##  rename "test.gds.tmp" (size: 2243).
##  # of fragments in total: 20.
```

```
# Summary
snpgdsSummary("test.gds")
```

```
## The file name: /tmp/RtmpqtRuP6/Rbuild14db5253eb32/SNPRelate/vignettes/test.gds
## The total number of samples: 3
## The total number of SNPs: 2
```

```
## SNP genotypes are stored in SNP-major mode (Sample X SNP).
```

3 Data Analysis

We developed *gdsfmt* and *SNPRelate* (high-performance computing R packages for multi-core symmetric multiprocessing computer architectures) to accelerate two key computations in GWAS: principal component analysis (PCA) and relatedness analysis using identity-by-descent (IBD) measures.

```
# Open the GDS file
genofile <- snpgdsOpen(snpgdsExampleFileName())
```

```
# Get population information
#   or pop_code <- scan("pop.txt", what=character())
#   if it is stored in a text file "pop.txt"
pop_code <- read.gdsn(index.gdsn(genofile, path="sample.annot/pop.group"))

# Display the first six values
head(pop_code)
```

```
## [1] "YRI" "YRI" "YRI" "YRI" "CEU" "CEU"
```

3.1 LD-based SNP pruning

It is suggested to use a pruned set of SNPs which are in approximate linkage equilibrium with each other to avoid the strong influence of SNP clusters in principal component analysis and relatedness analysis.

```
set.seed(1000)

# Try different LD thresholds for sensitivity analysis
snpset <- snpgdsLDpruning(genofile, ld.threshold=0.2)
```

```
## SNP pruning based on LD:
## Removing 365 SNP(s) on non-autosomes
## Removing 1 SNP (monomorphic: TRUE, < MAF: NaN, or > missing rate: NaN)
## Working space: 279 samples, 8722 SNPs
##  Using 1 (CPU) core
##  Sliding window: 500000 basepairs, Inf SNPs
##  |LD| threshold: 0.2
## Chromosome 1: 75.42%, 540/716
## Chromosome 2: 72.24%, 536/742
## Chromosome 3: 74.71%, 455/609
## Chromosome 4: 73.31%, 412/562
## Chromosome 5: 77.03%, 436/566
```

```
## Chromosome 6: 75.58%, 427/565
## Chromosome 7: 75.42%, 356/472
## Chromosome 8: 71.31%, 348/488
## Chromosome 9: 77.88%, 324/416
## Chromosome 10: 74.33%, 359/483
## Chromosome 11: 77.40%, 346/447
## Chromosome 12: 76.81%, 328/427
## Chromosome 13: 75.58%, 260/344
## Chromosome 14: 76.95%, 217/282
## Chromosome 15: 76.34%, 200/262
## Chromosome 16: 72.66%, 202/278
## Chromosome 17: 74.40%, 154/207
## Chromosome 18: 73.68%, 196/266
## Chromosome 19: 85.00%, 102/120
## Chromosome 20: 71.62%, 164/229
## Chromosome 21: 76.98%, 97/126
## Chromosome 22: 75.86%, 88/116
## 6547 SNPs are selected in total.
```

```
names(snpset)
```

```
##  [1] "chr1"  "chr2"  "chr3"  "chr4"  "chr5"  "chr6"  "chr7"  "chr8"  "chr9"  "chr10"
## [11] "chr11" "chr12" "chr13" "chr14" "chr15" "chr16" "chr17" "chr18" "chr19" "chr20"
## [21] "chr21" "chr22"
```

```
head(snpset$chr1)  # snp.id
```

```
## [1]  1  2  4  5  7 10
```

```
# Get all selected snp id
snpset.id <- unlist(snpset)
```

3.2 Principal Component Analysis

The functions in *SNPRelate* for PCA include calculating the genetic covariance matrix from genotypes, computing the correlation coefficients between sample loadings and genotypes for each SNP, calculating SNP eigenvectors (loadings), and estimating the sample loadings of a new dataset from specified SNP eigenvectors.

```
# Run PCA
pca <- snpgdsPCA(genofile)
```

```
## Principal Component Analysis (PCA) on SNP genotypes:
## Removing 365 SNP(s) on non-autosomes
## Removing 1 SNP (monomorphic: TRUE, < MAF: NaN, or > missing rate: NaN)
## Working space: 279 samples, 8722 SNPs
```

```
##  Using 1 (CPU) core
## PCA: the sum of all working genotypes (0, 1 and 2) = 2446510
## PCA: Mon Jan 19 21:04:52 2015 0%
## PCA: Mon Jan 19 21:04:53 2015 100%
## PCA: Mon Jan 19 21:04:53 2015 Begin (eigenvalues and eigenvectors)
## PCA: Mon Jan 19 21:04:53 2015 End (eigenvalues and eigenvectors)
```

The code below shows how to calculate the percent of variation is accounted for by the principal component for the first 16 PCs. It is clear to see the first two eigenvectors hold the largest percentage of variance among the population, although the total variance accounted for is still less the one-quarter of the total.

```
# variance proportion (%)
pc.percent <- pca$varprop*100
head(round(pc.percent, 2))
```

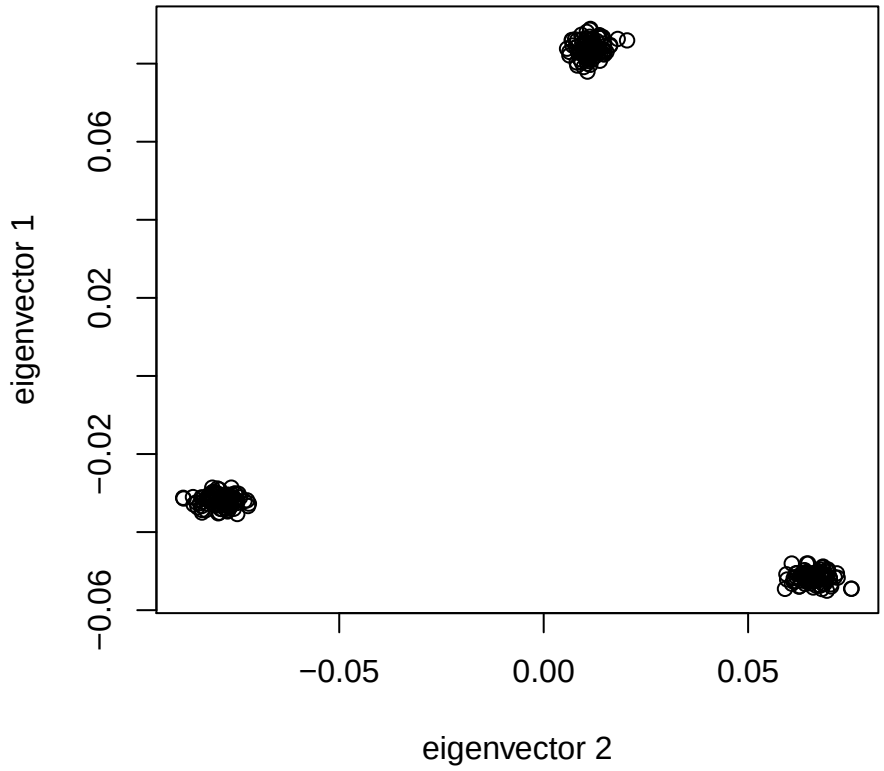
```
## [1] 12.23  5.84  1.01  0.95  0.84  0.74
```

In the case of no prior population information,

```
# make a data.frame
tab <- data.frame(sample.id = pca$sample.id,
    EV1 = pca$eigenvect[,1],    # the first eigenvector
    EV2 = pca$eigenvect[,2],    # the second eigenvector
    stringsAsFactors = FALSE)
head(tab)
```

```
##   sample.id         EV1         EV2
## 1   NA19152  0.08411287  0.01226860
## 2   NA19139  0.08360644  0.01085849
## 3   NA18912  0.08110808  0.01184524
## 4   NA19160  0.08680864  0.01447106
## 5   NA07034 -0.03109761 -0.07709255
## 6   NA07055 -0.03228450 -0.08155730
```

```
# Draw
plot(tab$EV2, tab$EV1, xlab="eigenvector 2", ylab="eigenvector 1")
```



If there are population information,

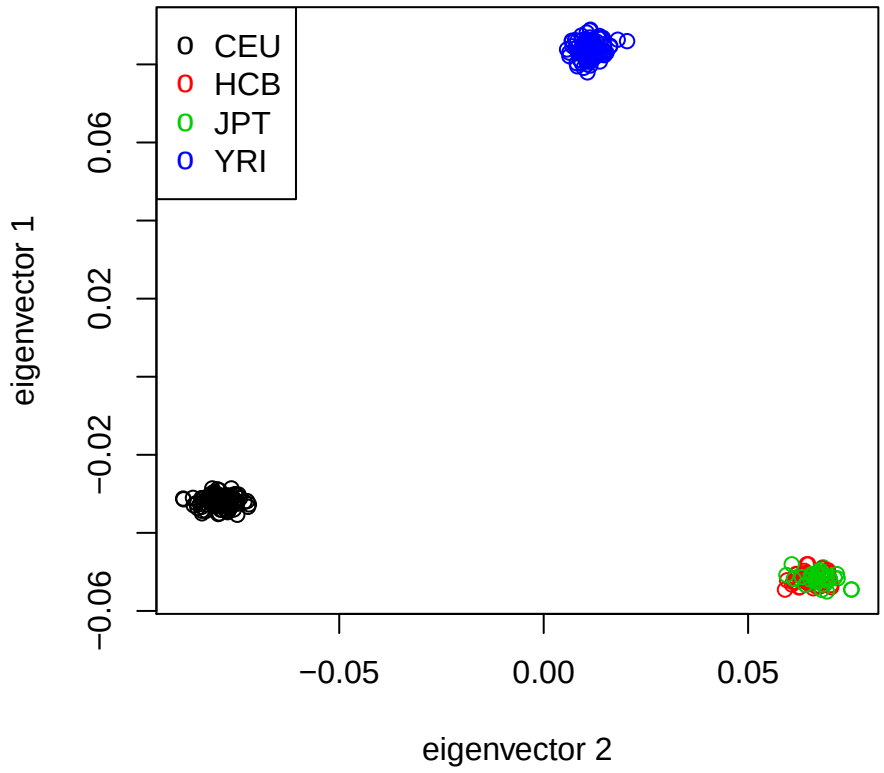
```
# Get sample id
sample.id <- read.gdsn(index.gdsn(genofile, "sample.id"))

# Get population information
#   or pop_code <- scan("pop.txt", what=character())
#   if it is stored in a text file "pop.txt"
pop_code <- read.gdsn(index.gdsn(genofile, "sample.annot/pop.group"))

# assume the order of sample IDs is as the same as population codes
head(cbind(sample.id, pop_code))
```

```
##      sample.id pop_code
## [1,] "NA19152" "YRI"
## [2,] "NA19139" "YRI"
## [3,] "NA18912" "YRI"
## [4,] "NA19160" "YRI"
## [5,] "NA07034" "CEU"
## [6,] "NA07055" "CEU"
```

```
# Make a data.frame
tab <- data.frame(sample.id = pca$sample.id,
    pop = factor(pop_code)[match(pca$sample.id, sample.id)],
    EV1 = pca$eigenvect[,1],    # the first eigenvector
    EV2 = pca$eigenvect[,2],    # the second eigenvector
    stringsAsFactors = FALSE)
head(tab)
```

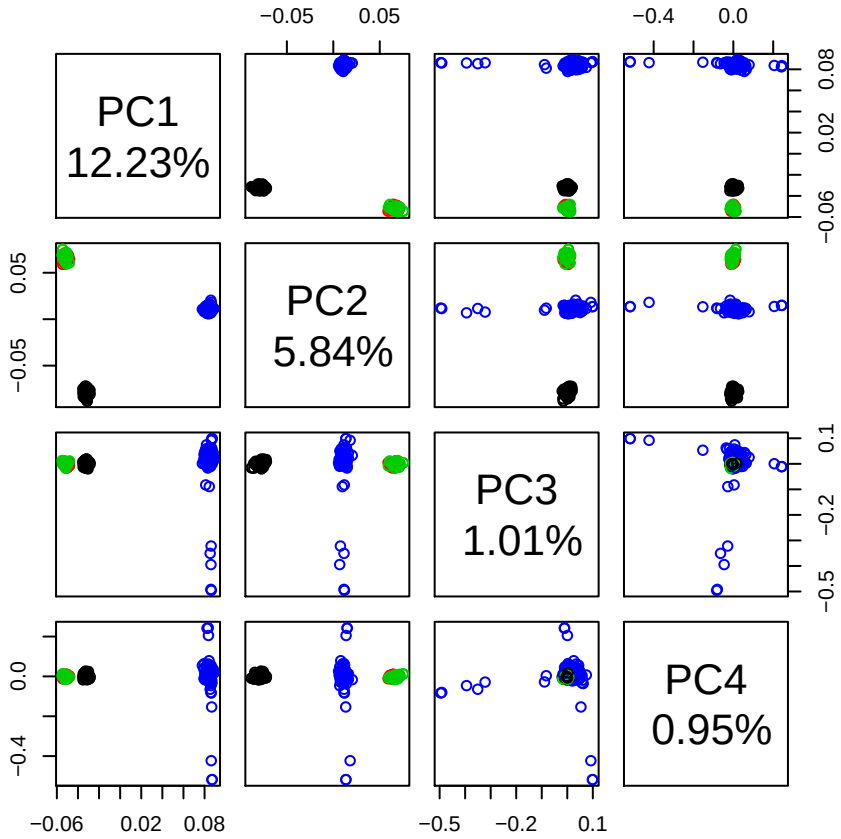
```
##   sample.id pop         EV1         EV2
## 1   NA19152 YRI  0.08411287  0.01226860
## 2   NA19139 YRI  0.08360644  0.01085849
## 3   NA18912 YRI  0.08110808  0.01184524
## 4   NA19160 YRI  0.08680864  0.01447106
## 5   NA07034 CEU -0.03109761 -0.07709255
## 6   NA07055 CEU -0.03228450 -0.08155730
```

```
# Draw
plot(tab$EV2, tab$EV1, col=as.integer(tab$pop),
    xlab="eigenvector 2", ylab="eigenvector 1")
legend("topleft", legend=levels(tab$pop), pch="o", col=1:nlevels(tab$pop))
```



Plot the principal component pairs for the first four PCs:

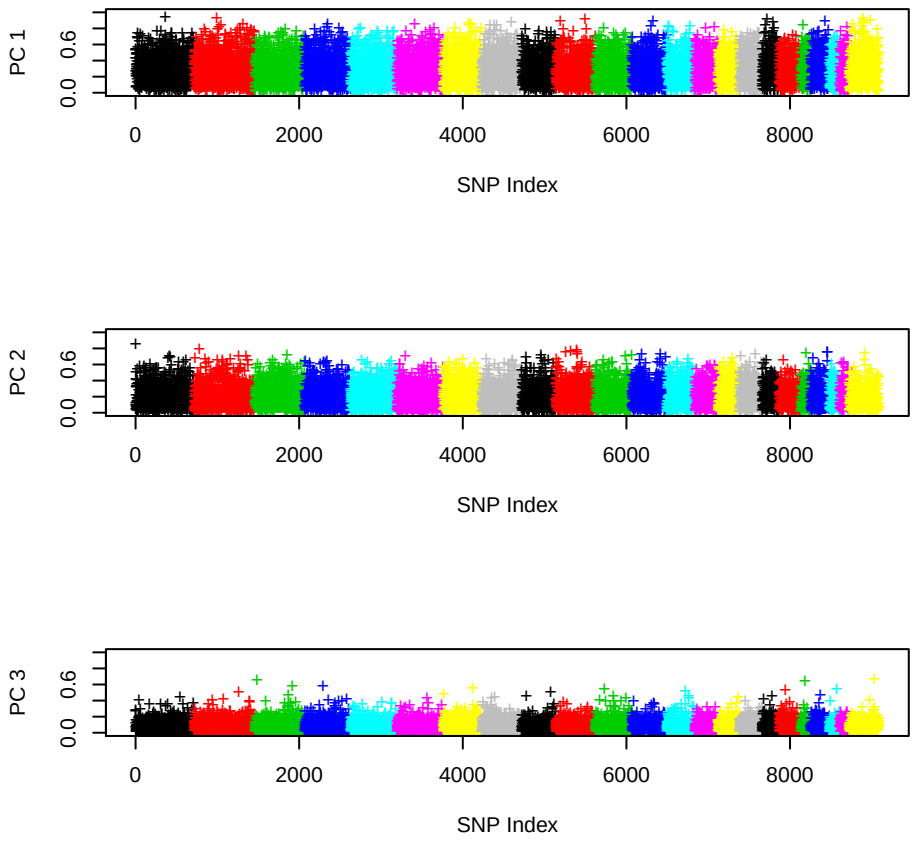
```
lbls <- paste("PC", 1:4, "\n", format(pc.percent[1:4], digits=2), "%", sep="")
pairs(pca$eigenvect[,1:4], col=tab$pop, labels=lbls)
```



To calculate the SNP correlations between eigenvactors and SNP genotypes:

```
# Get chromosome index
chr <- read.gdsn(index.gdsn(genofile, "snp.chromosome"))
CORR <- snpgdsPCACorr(pca, genofile, eig.which=1:4)

## SNP correlations:
## Working space: 279 samples, 9088 SNPs
##  Using 1 (CPU) core.
##  Using the top 32 eigenvectors.
## SNP Correlation: the sum of all working genotypes (0, 1 and 2) = 2553065
## SNP Correlation: Mon Jan 19 21:04:53 2015 0%
## SNP Correlation: Mon Jan 19 21:04:53 2015 100%

par( mfrow=c(3,1))
for (i in 1:3)
```

```
{
    plot(abs(CORR$snpcorr[i,]), ylim=c(0,1), xlab="SNP Index",
        ylab=paste("PC", i), col=chr, pch="+")
}
```



3.3 Relatedness Analysis

For relatedness analysis, identity-by-descent (IBD) estimation in *SNPRelate* can be done by either the method of moments (MoM) (Purcell et al., 2007) or maximum likelihood estimation (MLE) (Milligan, 2003; Choi et al., 2009). Although MLE estimates are more reliable than MoM, MLE is significantly more computationally intensive. For both of these methods it is preffered to use a LD pruned SNP set.

```
# YRI samples
sample.id <- read.gdsn(index.gdsn(genofile, "sample.id"))
YRI.id <- sample.id[pop_code == "YRI"]
```

3.3.1 Estimating IBD Using PLINK method of moments (MoM)

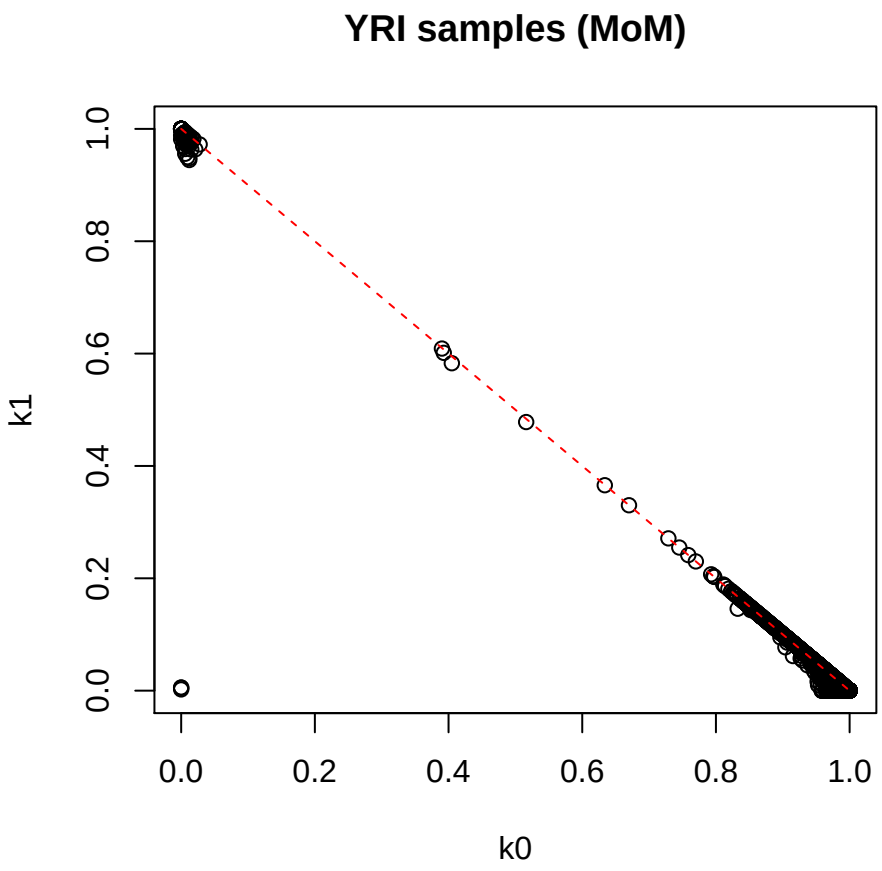
```
# Estimate IBD coefficients
ibd <- snpgdsIBDMoM(genofile, sample.id=YRI.id, snp.id=snpset.id,
    maf=0.05, missing.rate=0.05)
```

```
## IBD analysis (PLINK method of moment) on SNP genotypes:
## Removing 2541 SNP(s) on non-autosomes
## Removing 1285 SNPs (monomorphic: TRUE, < MAF: 0.05, or > missing rate: 0.05)
## Working space: 93 samples, 5262 SNPs
##  Using 1 (CPU) core
## PLINK IBD: the sum of all working genotypes (0, 1 and 2) = 484520
## PLINK IBD: Mon Jan 19 21:04:53 2015 0%
## PLINK IBD: Mon Jan 19 21:04:53 2015 100%
```

```
# Make a data.frame
ibd.coeff <- snpgdsIBDSelection(ibd)
head(ibd.coeff)
```

```
##       ID1     ID2        k0         k1     kinship
## 1 NA19152 NA19139 0.9548539 0.04514610 0.011286524
## 2 NA19152 NA18912 1.0000000 0.00000000 0.000000000
## 3 NA19152 NA19160 1.0000000 0.00000000 0.000000000
## 4 NA19152 NA18515 0.9234541 0.07654590 0.019136475
## 5 NA19152 NA19222 1.0000000 0.00000000 0.000000000
## 6 NA19152 NA18508 0.9833803 0.01661969 0.004154922
```

```
plot(ibd.coeff$k0, ibd.coeff$k1, xlim=c(0,1), ylim=c(0,1),
    xlab="k0", ylab="k1", main="YRI samples (MoM)")
lines(c(0,1), c(1,0), col="red", lty=2)
```

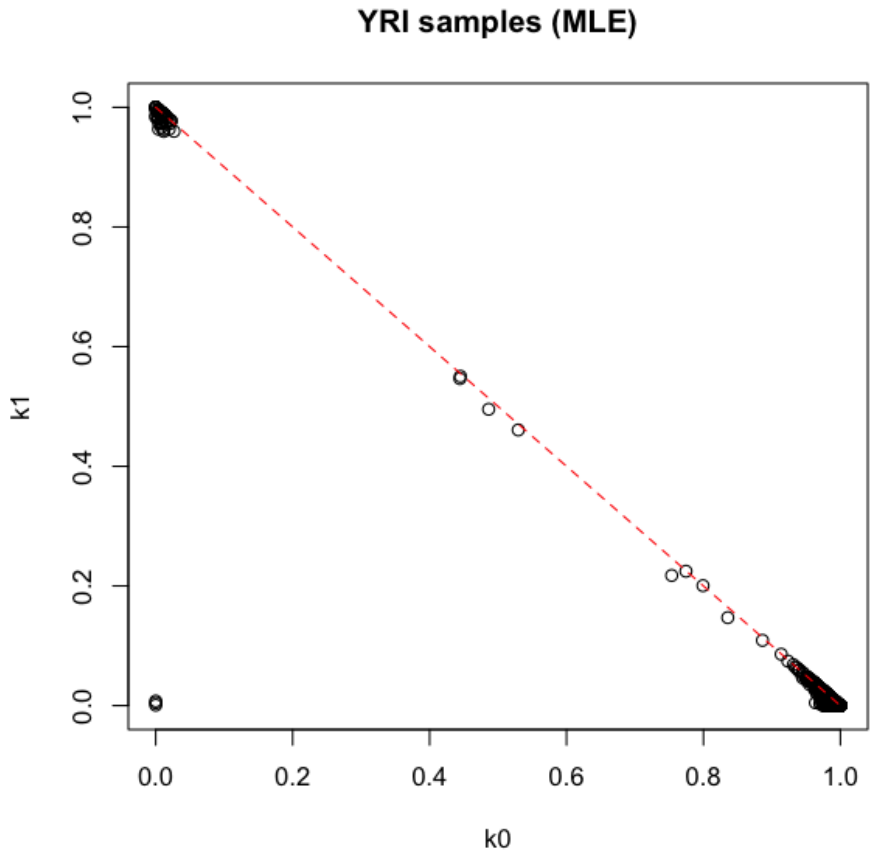


3.3.2 Estimating IBD Using Maximum Likelihood Estimation (MLE)

```
# Estimate IBD coefficients
set.seed(1000)
snp.id <- sample(snpset.id, 5000)  # random 5000 SNPs
ibd <- snpgdsIBDMLE(genofile, sample.id=YRI.id, snp.id=snp.id,
    maf=0.05, missing.rate=0.05)
```

```
# Make a data.frame
ibd.coeff <- snpgdsIBDSelection(ibd)
```

```
plot(ibd.coeff$k0, ibd.coeff$k1, xlim=c(0,1), ylim=c(0,1),
    xlab="k0", ylab="k1", main="YRI samples (MLE)")
lines(c(0,1), c(1,0), col="red", lty=2)
```



3.3.3 Relationship inference Using KING method of moments

Within- and between-family relationship could be inferred by the KING-robust method in the presence of population stratification.

```
# Incorporate with pedigree information
family.id <- read.gdsn(index.gdsn(genofile, "sample.annot/family.id"))
family.id <- family.id[match(YRI.id, sample.id)]
table(family.id)
```

```
## family.id
## 101 105 112 117  12  13  16  17  18  23  24  28   4  40  42  43  45  47  48   5  50  51
##   3   3   3   4   4   3   3   3   3   3   4   3   3   3   3   3   3   3   3   3   3   3
##  56  58  60  71  72  74  77   9
##   3   3   3   3   3   3   3   3
```

```
ibd.robust <- snpgdsIBDKING(genofile, sample.id=YRI.id, family.id=family.id)
```

```
## IBD analysis (KING method of moment) on SNP genotypes:
## Removing 365 SNP(s) on non-autosomes
## Removing 563 SNPs (monomorphic: TRUE, < MAF: NaN, or > missing rate: NaN)
```

```
## Working space: 93 samples, 8160 SNPs
##  Using 1 (CPU) core
## # of families: 30, and within- and between-family relationship are estimated differently
## Relationship inference in the presence of population stratification.
## KING IBD: the sum of all working genotypes (0, 1 and 2) = 755648
## KING IBD: Mon Jan 19 21:04:54 2015 0%
## KING IBD: Mon Jan 19 21:04:54 2015 100%
```

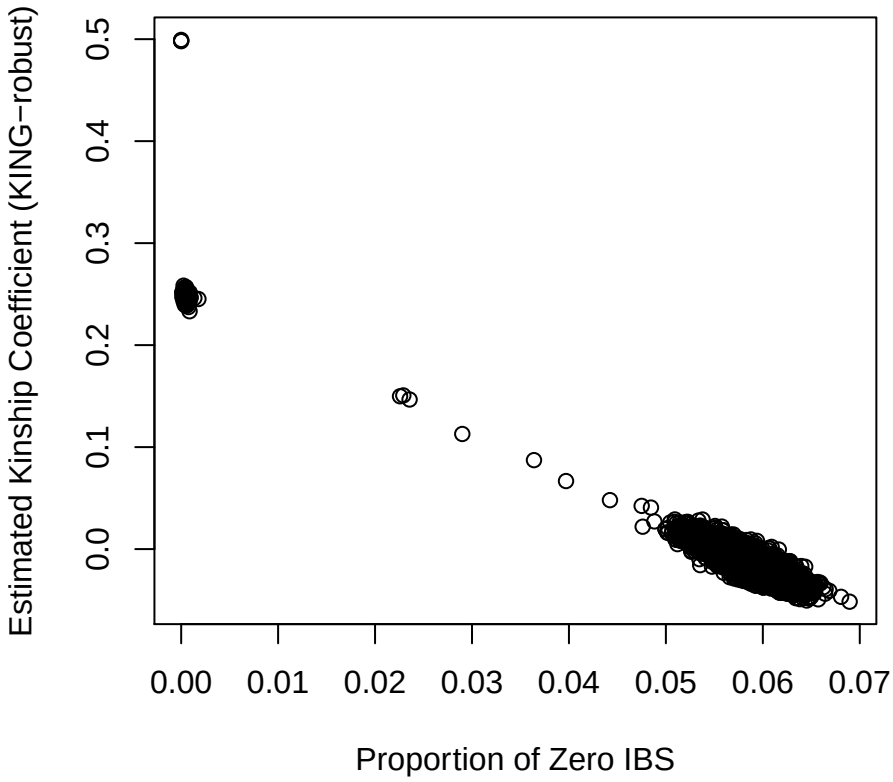
```
names(ibd.robust)
```

```
## [1] "sample.id" "snp.id"    "afreq"     "IBS0"      "kinship"
```

```
# Pairs of individuals
dat <- snpgdsIBDSelection(ibd.robust)
head(dat)
```

```
##       ID1     ID2       IBS0      kinship
## 1 NA19152 NA19139 0.05504926 -0.005516960
## 2 NA19152 NA18912 0.05738916 -0.003658537
## 3 NA19152 NA19160 0.06230760 -0.034086156
## 4 NA19152 NA18515 0.05602758  0.007874016
## 5 NA19152 NA19222 0.05923645 -0.012668574
## 6 NA19152 NA18508 0.05561722  0.002216848
```

```
plot(dat$IBS0, dat$kinship, xlab="Proportion of Zero IBS",
    ylab="Estimated Kinship Coefficient (KING-robust)")
```



3.4 Identity-By-State Analysis

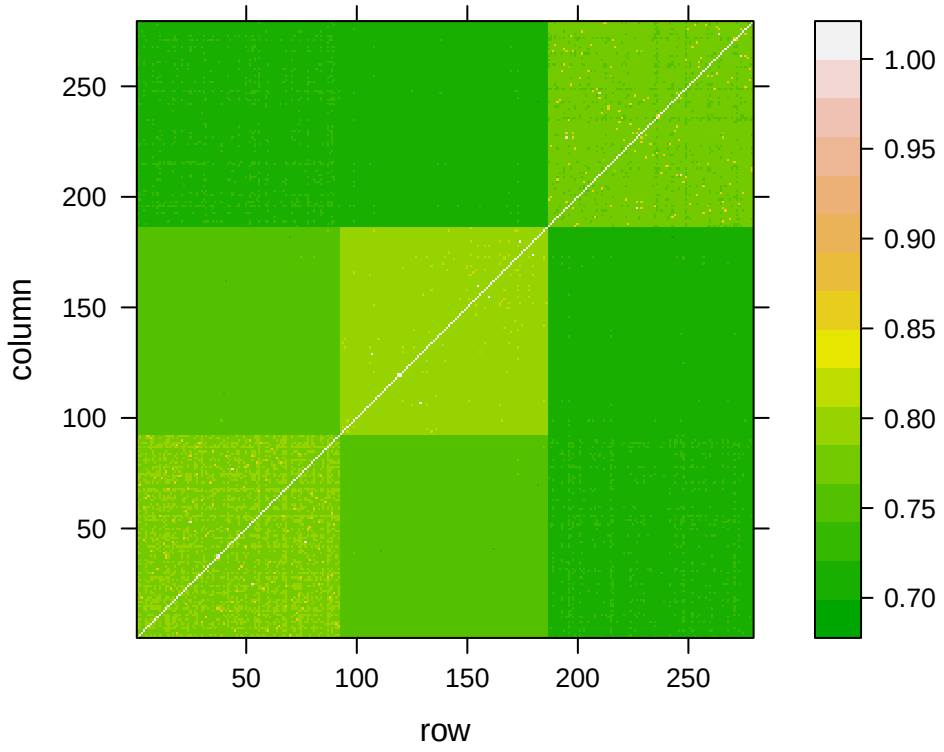
For the n individuals in a sample, `snpgdsIBS` can be used to create a $n \times n$ matrix of genome-wide average IBS pairwise identities:

```
ibs <- snpgdsIBS(genofile, num.thread=2)
```

```
## Identity-By-State (IBS) analysis on SNP genotypes:
## Removing 365 SNP(s) on non-autosomes
## Removing 1 SNP (monomorphic: TRUE, < MAF: NaN, or > missing rate: NaN)
## Working space: 279 samples, 8722 SNPs
##  Using 2 (CPU) cores
## IBS: the sum of all working genotypes (0, 1 and 2) = 2446510
## IBS: Mon Jan 19 21:04:54 2015 0%
## IBS: Mon Jan 19 21:04:54 2015 100%
```

The heat map is shown:

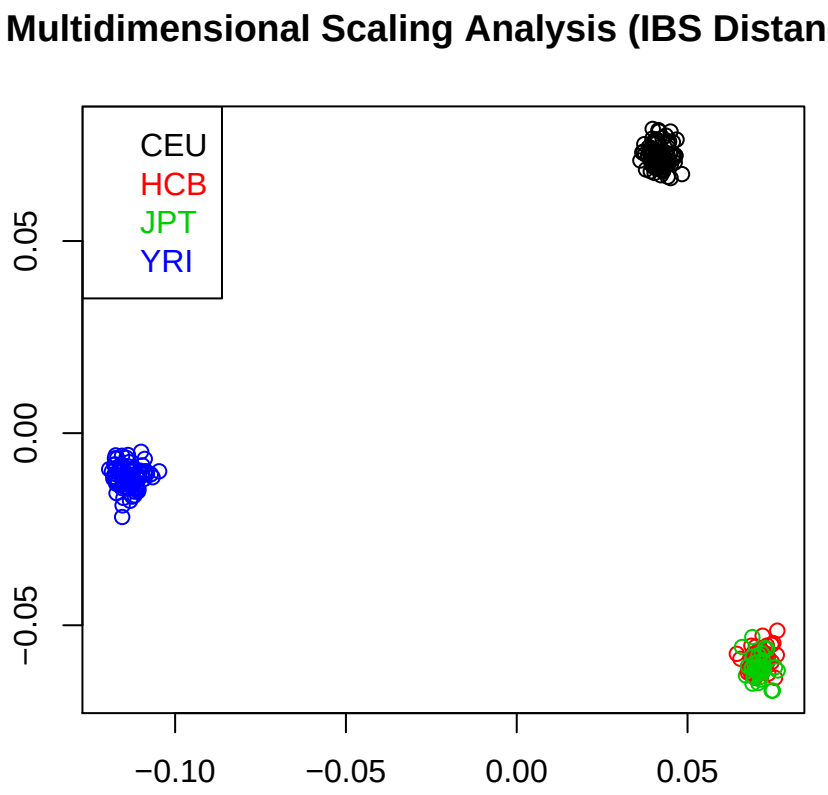
```
library(lattice)

L <- order(pop_code)
levelplot(ibs$ibs[L, L], col.regions = terrain.colors)
```



To perform multidimensional scaling analysis on the $n \times n$ matrix of genome-wide IBS pairwise distances:

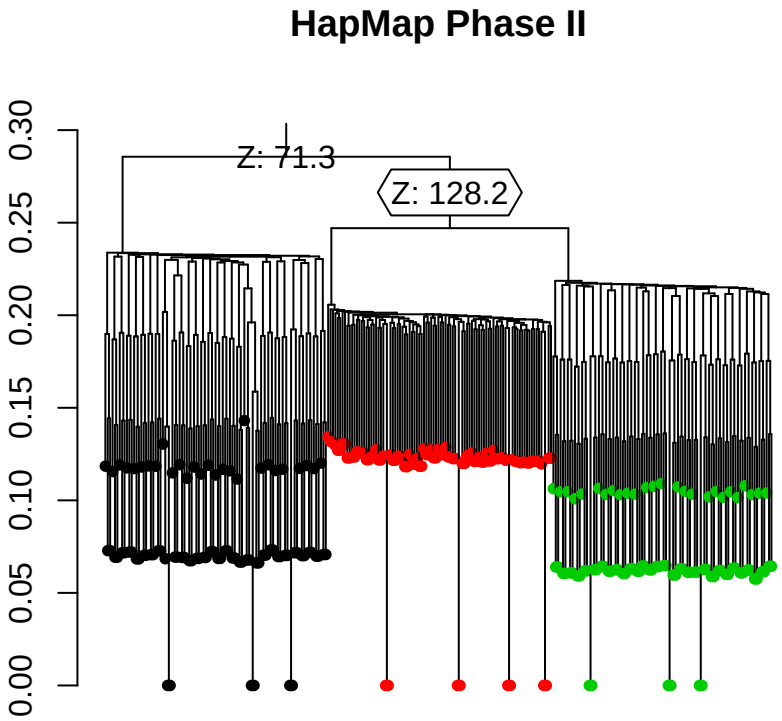
```
loc <- cmdscale(1 - ibs$ibs, k = 2)
x <- loc[, 1]; y <- loc[, 2]
race <- as.factor(pop_code)
```

```
plot(x, y, col=race, xlab = "", ylab = "",
    main = "Multidimensional Scaling Analysis (IBS Distance)")
legend("topleft", legend=levels(race), text.col=1:nlevels(race))
```



To perform cluster analysis on the $n \times n$ matrix of genome-wide IBS pairwise distances, and determine the groups by a permutation score:

```
set.seed(100)
ibs.hc <- snpgdsHCluster(snpgdsIBS(genofile, num.thread=2))

## Identity-By-State (IBS) analysis on SNP genotypes:
## Removing 365 SNP(s) on non-autosomes
## Removing 1 SNP (monomorphic: TRUE, < MAF: NaN, or > missing rate: NaN)
## Working space: 279 samples, 8722 SNPs
##  Using 2 (CPU) cores
## IBS: the sum of all working genotypes (0, 1 and 2) = 2446510
## IBS: Mon Jan 19 21:04:55 2015 0%
## IBS: Mon Jan 19 21:04:55 2015 100%

# Determine groups of individuals automatically
rv <- snpgdsCutTree(ibs.hc)

## Determine groups by permutation (Z threshold: 15, outlier threshold: 5):
## Create 3 groups.

plot(rv$dendrogram, leaflab="none", main="HapMap Phase II")
```




```
table(rv$samp.group)
```

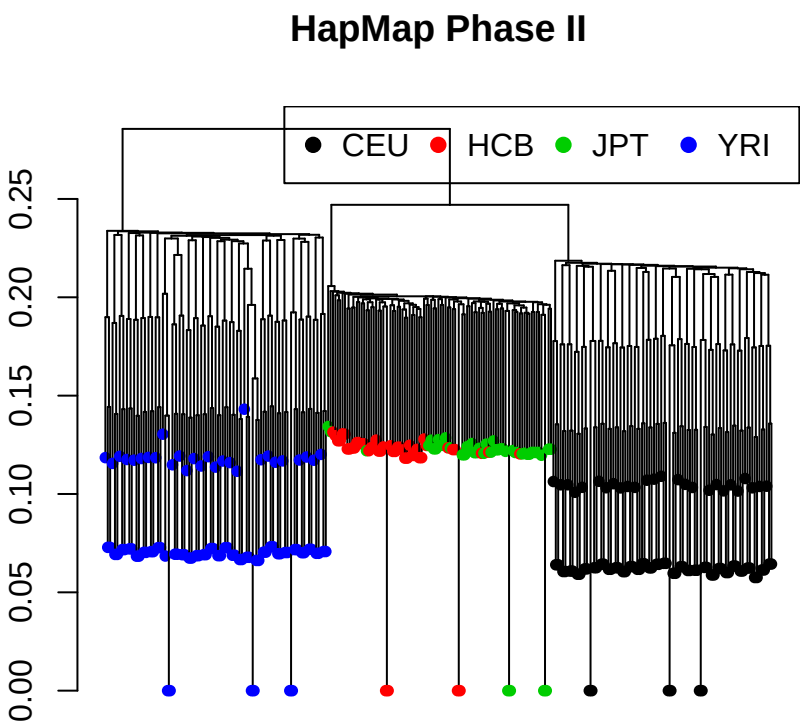
```
##
## G001 G002 G003
##   93   94   92
```

Here is the population information we have known:

```
# Determine groups of individuals by population information
rv2 <- snpgdsCutTree(ibs.hc, samp.group=as.factor(pop_code))
```

```
## Create 4 groups.
```

```
plot(rv2$dendrogram, leaflab="none", main="HapMap Phase II")
legend("topright", legend=levels(race), col=1:nlevels(race), pch=19, ncol=4)
```




```
# Close the GDS file
snpgdsClose(genofile)
```

4 Resources

1. *CoreArray* project: http://corearray.sourceforge.net/
2. *gdsfmt* R package: https://github.com/zhengxwen/gdsfmt
3. *SNPRelate* R package: http://www.bioconductor.org/packages/devel/bioc/html/SNPRelate.html
4. GENEVA software: https://www.genevastudy.org/Accomplishments/software
5. *GWASTools*: an R/Bioconductor package for quality control and analysis of Genome-Wide Association Studies http://www.bioconductor.org/packages/devel/bioc/html/GWASTools.html

5 Session Info

```
toLatex(sessionInfo())
```

- R version 3.1.2 (2014-10-31), x86_64-unknown-linux-gnu
- Locale: LC_CTYPE=en_US.UTF-8, LC_NUMERIC=C, LC_TIME=en_US.UTF-8, LC_COLLATE=C, LC_MONETARY=en_US.UTF-8, LC_MESSAGES=en_US.UTF-8, LC_PAPER=en_US.UTF-8, LC_NAME=C, LC_ADDRESS=C, LC_TELEPHONE=C, LC_MEASUREMENT=en_US.UTF-8, LC_IDENTIFICATION=C
- Base packages: base, datasets, grDevices, graphics, methods, stats, utils
- Other packages: SNPRelate 1.0.1, gdsfmt 1.2.2, lattice 0.20-29
- Loaded via a namespace (and not attached): BiocStyle 1.4.1, evaluate 0.5.5, formatR 1.0, grid 3.1.2, highr 0.4, knitr 1.8, stringr 0.6.2, tools 3.1.2

6 References

1. **A High-performance Computing Toolset for Relatedness and Principal Component Analysis of SNP Data.** Xiuwen Zheng; David Levine; Jess Shen; Stephanie M. Gogarten; Cathy Laurie; Bruce S. Weir. Bioinformatics 2012; doi: 10.1093/bioinformatics/bts606.
2. **GWASTools: an R/Bioconductor package for quality control and analysis of Genome-Wide Association Studies.** Stephanie M. Gogarten, Tushar Bhangale, Matthew P. Conomos, Cecelia A. Laurie, Caitlin P. McHugh, Ian Painter, Xiuwen Zheng, David R. Crosslin, David Levine, Thomas Lumley, Sarah C. Nelson, Kenneth Rice, Jess Shen, Rohit Swarnkar, Bruce S. Weir, and Cathy C. Laurie. Bioinformatics 2012; doi:10.1093/bioinformatics/bts610.
3. **Quality control and quality assurance in genotypic data for genome-wide association studies.** Laurie CC, Doheny KF, Mirel DB, Pugh EW, Bierut LJ, Bhangale T, Boehm F, Caporaso NE, Cornelis MC, Edenberg HJ, Gabriel SB, Harris EL, Hu FB, Jacobs KB, Kraft P, Landi MT, Lumley T, Manolio TA, McHugh C, Painter I, Paschall J, Rice JP, Rice KM, Zheng X, Weir BS; GENEVA Investigators. Genet Epidemiol. 2010 Sep;34(6):591-602.

7 Acknowledgements

The author would like to thank members of the GENEVA consortium (http://www.genevastudy.org) for access to the data used for testing the *gdsfmt* and *SNPRelate* packages.