Practical in Polygenic Risk Scoring using the R package qgg

Palle Duun Rohde & Peter Sørensen

2022-10-07

Introduction

The aim of this practical is to provide a simple introduction to polygenic risk scoring (PRS) of complex traits and diseases. The practical will be a mix of theoretical and practical exercises in R that are used for illustrating/applying the theory presented in the corresponding lecture on polygenic risk scoring:

- Data used for computing polygenic risk scores
- Methods used for computing polygenic risk scores
- Methods used for evaluating the predictive ability of the polygenic risk scores

Sessions: This practical provides a step-by-step guide to performing basic PRS analyses including the following sessions:

- Session 1: Use R for downloading data
- Session 2: Prepare and explore phenotype data
- Session 3: Prepare and perform quality control of genetic data
- Session 4: Compute GWAS summary statistics
- Session 5: Compute sparse LD matrices
- Session 6: Compute PRS using clumping and thresholding (C+T)
- Session 7: Compute PRS using different Bayesian Linear Regression (BLR) models

Polygenic risk scores: Polygenic risk scoring combines information from large numbers of markers across the genome (hundreds to millions) to give a single numerical score for an individual's relative risk for developing a specific disease on the basis of the DNA variants they have inherited.

For a particular disease or trait a polygenic risk score (PRS) is calculated as:

$$PRS = \sum_{i=1}^{m} X_i \hat{b}_i$$

where X_i is the genotype vector, and $\hat{b}_i$ the weight of the i'th single genetic marker.

Genomic prediction has been used for many years in animal and plant breeding (e.g., Meuwissen et al. 2001), and genomic prediction (i.e., PRS) has gained popularity during the last decade because of:

- Larger GWAS sample size = more precision for effect estimates
- Development of methods that combine genome-wide sets of variants
- Large biobanks for validation and testing of genetic risk scores
- Ability to identify clinically meaningful increases in disease risk predictions

Terminology: Polygenic risk scores, polygenic scores, genomic risk score, genetic scores, genetic predispostion, genetic value, genomic breeding value is (more or less) the same thing.

Complex traits and diseases: For many complex traits and diseases there will be thousands of genetic variants that each contribute with a small effect on the disease risk or quantitative trait. Rare variant with large effects will only explain small proportion of h^2 (low predictive potential). Common variants with small effects can explain larger proportion of h^2 (high predictive potential). The majority of complex traits and common diseases in humans are heritable. The heritability determines the value of using genetics for risk prediction. In general, large data sets are required to obtain accurate marker estimates of small to moderate effects, which also improves the prediction accuracy.

Heritability: The heritability (h^2) quantify the degree of variation in a phenotypic trait in a population that is due to genetic variation between individuals in that population. It measures how much of the variation of a trait can be attributed to variation of genetic factors, as opposed to variation of environmental factors. The narrow sense heritability is the ratio of additive genetic variance (σ^2_a) to the overall phenotypic variance ($\sigma^2_y = \sigma^2_a + \sigma^2_e$):

$$h^2 = \sigma^2_a/(\sigma^2_a + \sigma^2_e) \tag{1}$$

A heritability of 0 implies that no genetic effects influence the observed variation in the trait, while a heritability of 1 implies that all of the variation in the trait is explained by the genetic effects. In general, the amount of information provided by the phenotype about the genetic risk is determined by the heritability. Note that heritability is population-specific and a heritability of 0 does not necessarily imply that there is no genetic determinism for the trait.

Software used: To follow the practical, you will need the following installed (see installation guides below):

- R (version $\geq$ 4.2)
- qgg (version $\geq$ 1.1.1)

We assume you have basic knowledge on how to use R. We suggest to use R through the user-friendly interface called Rstudio (although this is not a requirement).

Install R and Rstudio

R is a free software environment for statistical computing and graphics (https://www.r-project.org/). Because R is free and it is available for the most commonly used operating systems such as Windows, MacOSX and Linux, it has become very popular in statistics and in data science. Furthermore, R can be extended with user-contributed code and documentation (called R-packages) in a very easy and standardised way. The number of available R-packages is growing rapidly and has reached more than 18000 (https://cran.r-project.org/web/packages/).

RStudio (https://www.rstudio.com/) is a private company that offers a number of different products, with one being Rstudio which is an Integrated Development Environment (IDE) for R. A great number of different resources about R and RStudio IDE is available.

Install R from here: https://mirrors.dotsrc.org/cran/

Install Rstudio (free version) from here: https://www.rstudio.com/products/rstudio/download/

Further information and introduction to R and Rstudio can be found here:

https://cran.r-project.org/doc/manuals/r-release/R-intro.html

https://www.rstudio.com/resources/cheatsheets

https://www.rstudio.com/resources/webinars

Linking R to multi-threaded math libraries (DO NOT INSTALL IN THIS PRACTICAL)

The multi-core machines of today offer parallel processing power. To take advantage of this, R should be linked to multi-threaded math libraries (e.g. MKL/ OpenBLAS/ATLAS). These libraries make it possible for so many common R operations, such as matrix multiply/inverse, matrix decomposition, and some higher-level matrix operations, to compute in parallel and use all of the processing power available to reduce computation times.

This can make a huge difference in computation times: https://mran.microsoft.com/documents/rro/multithread#mt-bench

For Windows/Linux users it is possible to install Microsoft R Open is the enhanced distribution of R from Microsoft Corporation: https://mran.microsoft.com/open

For MAC users the ATLAS (Automatically Tuned Linear Algebra Software) library can be installed from here: https://ports.macports.org/port/atlas/

Brief Introduction to the qgg R package

The practical is based on the R package `qgg` (Rohde et al. (2021, 2022)). This package provides an infrastructure for efficient processing of large-scale genetic and phenotypic data including core functions for:

- fitting linear mixed models
- constructing genetic relationship matrices
- estimating genetic parameters (heritability and correlation)
- performing genomic prediction and genetic risk profiling
- single or multi-marker association analyses

`qgg` handles large-scale data by taking advantage of:

- multi-core processing using openMP
- multithreaded matrix operations implemented in BLAS libraries (e.g., OpenBLAS, ATLAS or MKL)
- fast and memory-efficient batch processing of genotype data stored in binary files (i.e., PLINK bedfiles)

You can install qgg from CRAN with:

```
install.packages("qgg")
```

The most recent version of `qgg` can be obtained from github:

```
library(devtools)
devtools::install_github("psoerensen/qgg")
```

Input data/objects commonly used in the qgg package

All functions in `qgg` used for analysis of complex traits relies on a simple data infrastructure that takes the following main input:

`y`: vector, matrix or list of phenotypes
`X`: design matrix for non-genetic factors
`W`: matrix of centered and scaled genotypes (in memory)
`Glist`: list structure providing information on genotypes, sparse LD, and LD scores (on disk)
`stat`: data frame with marker summary statistics
`sets`: list of sets with marker ids
`ids`: vector of ids of individuals
`rsids`: vector marker marker ids

Session 1: Downloading the data using R

In this practical we will perform polygenic risk scoring based on simulated data. The data consist of disease phenotype, covariable, and genetic marker data. The data used in this practical are intended for demonstration purposes only.

Load required packages:

```
library(data.table)
library(tools)
```

Create (your own) directory for downloading files:

```
dir.create("C:\\Users\\au223366\\Dropbox\\Projects\\Summer_course")
```

Set (your own) working directory for the downloaded files:

```
setwd("C:\\Users\\au223366\\Dropbox\\Projects\\Summer_course")
```

Download PLINK genotype files (bedfile, bimfile, famfile) from github repository:

Genetic data are commonly stored in a binary format (as used by the software PLINK), named `.bed`-files. These files must be accompanied by `.bim` (contains information about the genetic variants) and `.fam` (contains information about the individuals) files. Read more about these file formats here:

1. https://www.cog-genomics.org/plink/1.9/formats#bed
2. https://www.cog-genomics.org/plink/1.9/formats#bim
3. https://www.cog-genomics.org/plink/1.9/formats#fam

```
url <- "https://github.com/psoerensen/qgdata/raw/main/simulated_human_data/human.bed"
download.file(url = url, mode = "wb", destfile = "human.bed")
url <- "https://github.com/psoerensen/qgdata/raw/main/simulated_human_data/human.bim"
download.file(url = url, destfile = "human.bim")
url <- "https://github.com/psoerensen/qgdata/raw/main/simulated_human_data/human.fam"
download.file(url = url, destfile = "human.fam")
```

Note that `mode="wb"` for downloading the human.bed file. This is needed or otherwise the bed-file will be corrupted. If the data file is corrupted it can cause errors in the analyses.

Check md5sum:

A md5sum hash is generally included in files so that file integrity can be checked. The following command performs this md5sum check in R:

```
md5sum("C:\\Users\\au223366\\Dropbox\\Projects\\Summer_course\\human.bed")
```

This should be compared to the md5sum value before download:

```
# for MacOS / Linux users
system(paste("curl -sL https://github.com/psoerensen/qgdata/raw/main/simulated_human_data
             /human.bed | md5"))
```

Read more about md5sum here: https://en.wikipedia.org/wiki/Md5sum

Download pheno and covar files from github repository;

```
url <- "https://github.com/psoerensen/qgdata/raw/main/simulated_human_data/human.pheno"
download.file(url = url, destfile = "human.pheno")
url <- "https://github.com/psoerensen/qgdata/raw/main/simulated_human_data/human.covar"
download.file(url = url, destfile = "human.covar")
```

Session 2: Preparing the phenotype and covariable data using R

One of the first thing to do is to prepare the phenotypic data used in the analysis. The goal is to understand the variables, how many records the data set contains, how many missing values, what is the variable structure, what are the variable relationships and more.

Several functions can be used (e.g., `str()`, `head()`, `dim()`, `table()`,`is.na()`).

```
library(data.table)
```

Read phenotype and covariables data files

```
pheno <- fread(input = "C:\\Users\\au223366\\Dropbox\\Projects\\Summer_course\\human.pheno",
    data.table = FALSE)
```

```
covar <- fread(input = "C:\\Users\\au223366\\Dropbox\\Projects\\Summer_course\\human.covar",
    data.table = FALSE)
```

How many observations and which variables do we have in the data set?

To get an overview of the data set you are working with you can use the `str()` or `head()` functions:

```
str(pheno)
```

```
## 'data.frame':    5000 obs. of  3 variables:
##  $ V1: chr  "IND1" "IND2" "IND3" "IND4" ...
##  $ V2: chr  "IND1" "IND2" "IND3" "IND4" ...
##  $ V3: int  0 0 1 0 0 1 0 0 1 1 ...
```

```
str(covar)
```

```
## 'data.frame':    5000 obs. of  14 variables:
##  $ V1 : chr  "IND1000" "IND1001" "IND1002" "IND1003" ...
##  $ V2 : chr  "IND1000" "IND1001" "IND1002" "IND1003" ...
##  $ V3 : int  1 0 0 0 0 1 1 1 1 0 ...
##  $ V4 : num  61.2 62.4 65.6 55.4 64.1 ...
##  $ V5 : num  0.00488 -0.0063 -0.00522 0.02028 -0.00411 ...
##  $ V6 : num  -0.002097 0.000148 -0.000606 0.006778 0.004349 ...
##  $ V7 : num  0.01091 0.00469 0.01709 -0.00306 -0.00839 ...
##  $ V8 : num  0.0056 0.01611 -0.00442 -0.01284 0.01022 ...
##  $ V9 : num  -0.008546 -0.00262 0.034218 -0.003078 -0.000912 ...
##  $ V10: num  0.019736 0.000308 -0.005157 0.012835 0.004788 ...
##  $ V11: num  -0.00408 0.00961 -0.00289 -0.01387 0.00977 ...
##  $ V12: num  0.01435 -0.01479 -0.00901 0.01499 0.00899 ...
##  $ V13: num  -0.00152 -0.00708 0.00706 0.00474 -0.00381 ...
##  $ V14: num  0.003998 0.037763 -0.000359 -0.022958 -0.010249 ...
```

```
head(pheno)
```

```
##     V1   V2 V3
## 1 IND1 IND1  0
## 2 IND2 IND2  0
## 3 IND3 IND3  1
## 4 IND4 IND4  0
## 5 IND5 IND5  0
## 6 IND6 IND6  1
```

```
head(covar)
```

```
##        V1      V2 V3       V4        V5        V6        V7        V8        V9
## 1 IND1000 IND1000  1 61.24445  0.004881 -0.002097  0.010908  0.005597 -0.008546
## 2 IND1001 IND1001  0 62.42271 -0.006301  0.000148  0.004692  0.016107 -0.002620
## 3 IND1002 IND1002  0 65.64274 -0.005215 -0.000606  0.017091 -0.004416  0.034218
## 4 IND1003 IND1003  0 55.40703  0.020284  0.006778 -0.003061 -0.012837 -0.003078
## 5 IND1004 IND1004  0 64.10666 -0.004105  0.004349 -0.008388  0.010221 -0.000912
## 6 IND1005 IND1005  1 65.77823 -0.010636 -0.021853 -0.017450  0.010058  0.000502
##         V10       V11       V12       V13       V14
## 1  0.019736 -0.004078  0.014350 -0.001518  0.003998
## 2  0.000308  0.009612 -0.014793 -0.007077  0.037763
## 3 -0.005157 -0.002894 -0.009013  0.007064 -0.000359
## 4  0.012835 -0.013872  0.014987  0.004744 -0.022958
## 5  0.004788  0.009773  0.008988 -0.003814 -0.010249
## 6  0.003807 -0.003007  0.005646 -0.007199 -0.009874
```

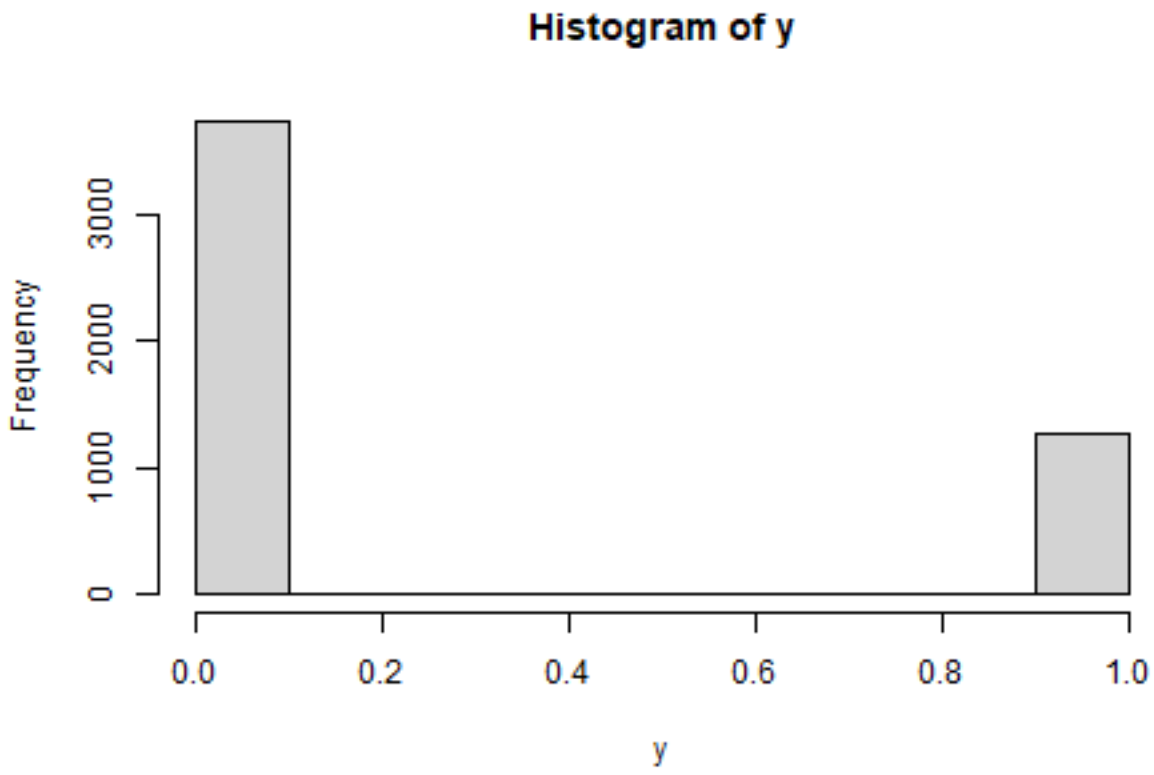
How is the phenotype distributed?

Define the response variable

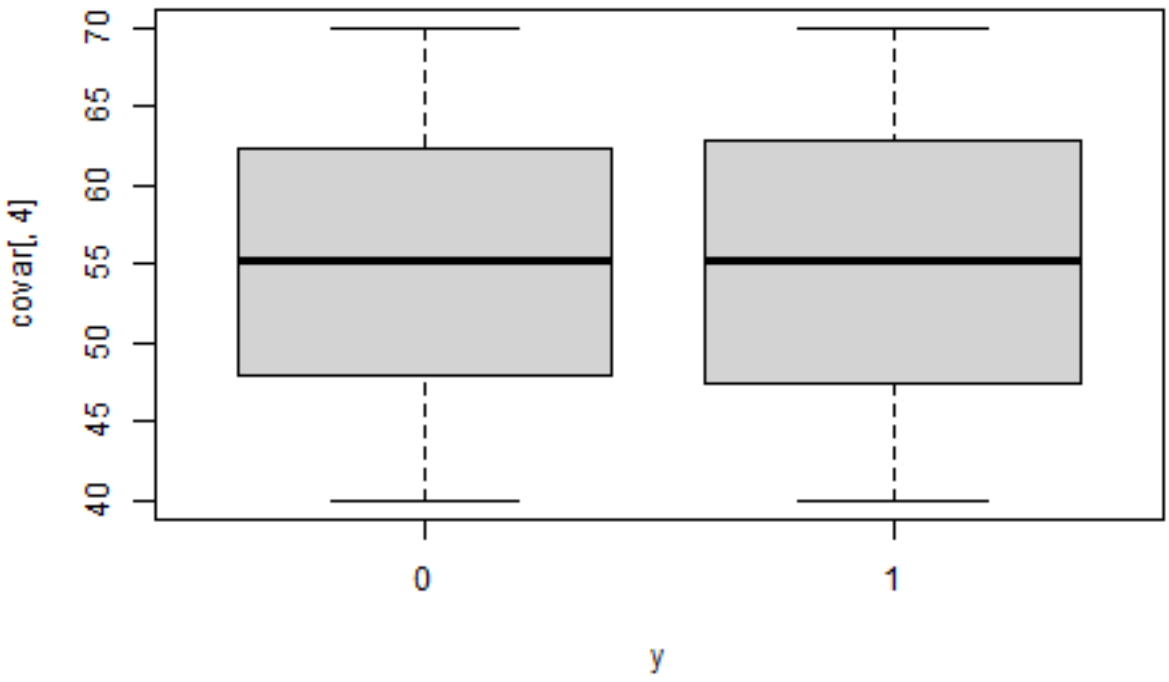
```
y <- pheno[, 3]
names(y) <- pheno[, 1]
```

Use the histogram and boxplot functions to visualize the distribution of the trait/covariables:

```
hist(y)
```




```
boxplot(covar[, 4] ~ y)
```



Which factors or covariated influence the phenotype?

The exploratory data analysis is the process of analyzing and visualizing the data to get a better understanding of the data. It is not a formal statistical test. Which factors should we include in the statistical model? To best answer these question we can fit a logistic regression model that include these factors in the model.

This can be done using the `glm()` function:

```
fit <- glm(y ~ V3 + V4 + V5 + V6 + V7 + V8 + V9 + V10 + V11 + V12 + V13 +
    V14, data = covar, family = binomial(link = "logit"))
summary(fit)
```

```
##
## Call:
## glm(formula = y ~ V3 + V4 + V5 + V6 + V7 + V8 + V9 + V10 + V11 +
##     V12 + V13 + V14, family = binomial(link = "logit"), data = covar)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -0.9283  -0.7765  -0.7439   1.5279   1.8371
##
## Coefficients:
##              Estimate Std. Error z value Pr(>|z|)
## (Intercept) -1.181957   0.214978  -5.498 3.84e-08 ***
```

```
## V3          -0.011066   0.065316  -0.169   0.8655
## V4           0.001740   0.003818   0.456   0.6487
## V5           1.309941   2.306100   0.568   0.5700
## V6          -0.280830   2.307468  -0.122   0.9031
## V7           2.354268   2.306546   1.021   0.3074
## V8          -1.349941   2.306494  -0.585   0.5584
## V9           5.520940   2.311308   2.389   0.0169 *
## V10         -2.724333   2.307482  -1.181   0.2377
## V11         -1.929366   2.307348  -0.836   0.4031
## V12          0.578919   2.308375   0.251   0.8020
## V13          4.382604   2.307287   1.899   0.0575 .
## V14         -0.504532   2.306778  -0.219   0.8269
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 5645.2  on 4999  degrees of freedom
## Residual deviance: 5631.6  on 4987  degrees of freedom
## AIC: 5657.6
##
## Number of Fisher Scoring iterations: 4
```

The exploration (including quality control) of phenotypes and covariables is a key step in quantitative genetic analyses. It is, however, beyond the scope of this practical.

Session 3: Prepare genotype for simulated data

The preparation (including quality control) of genotype data is a key step in quantitative genetic analyses.

```
library(qgg)
```

Summarize genotype information in PLINK files

The function `gprep()` reads genotype information from binary PLINK files, and creates the `Glist` object that contains general information about the genotypes:

```
bedfiles <- "C:\\Users\\au223366\\Dropbox\\Projects\\Summer_course\\human.bed"
bimfiles <- "C:\\Users\\au223366\\Dropbox\\Projects\\Summer_course\\human.bim"
famfiles <- "C:\\Users\\au223366\\Dropbox\\Projects\\Summer_course\\human.fam"

Glist <- gprep(study = "Example", bedfiles = bedfiles, bimfiles = bimfiles,
    famfiles = famfiles)
saveRDS(Glist, file = "C:\\Users\\au223366\\Dropbox\\Projects\\Summer_course\\Glist.RDS",
    compress = FALSE)
```

The output from `gprep()` (`Glist`) has a list structure that contains information about the genotypes in the binary file. `Glist` is required for downstream analyses provided in the qgg package. Typically, the `Glist` is prepared once, and saved as an *.RDS-file. To explore the content of the `Glist` object:

```
names(Glist)
```

```
##  [1] "study"    "bedfiles" "bimfiles" "famfiles" "ids"      "n"
##  [7] "rsids"    "mchr"     "a1"       "a2"       "position" "chr"
## [13] "cpra"     "map"      "nmiss"    "af"       "af1"      "af2"
## [19] "maf"      "hom"      "het"      "n0"       "n1"       "n2"
## [25] "nchr"
```

```
str(Glist)
```

```
## List of 25
##  $ study   : chr "Example"
##  $ bedfiles: chr "C:\\Users\\au223366\\Dropbox\\Projects\\Summer_course\\human.bed"
##  $ bimfiles: chr "C:\\Users\\au223366\\Dropbox\\Projects\\Summer_course\\human.bim"
##  $ famfiles: chr "C:\\Users\\au223366\\Dropbox\\Projects\\Summer_course\\human.fam"
##  $ ids     : chr [1:5000] "IND1000" "IND1001" "IND1002" "IND1003" ...
##  $ n       : int 5000
##  $ rsids   :List of 1
##   ..$ : Named chr [1:50000] "1:568670" "1:761732" "1:830791" "1:867663" ...
##   .. ..- attr(*, "names")= chr [1:50000] "1_568670_C_T" "1_761732_C_T" "1_830791_T_C" "1_867663_T_C"
##  $ mchr    : int 50000
##  $ a1      :List of 1
##   ..$ : Named chr [1:50000] "C" "C" "T" "T" ...
##   .. ..- attr(*, "names")= chr [1:50000] "1:568670" "1:761732" "1:830791" "1:867663" ...
##  $ a2      :List of 1
##   ..$ : Named chr [1:50000] "T" "T" "C" "C" ...
##   .. ..- attr(*, "names")= chr [1:50000] "1:568670" "1:761732" "1:830791" "1:867663" ...
```

```
##  $ position:List of 1
##   ..$ : Named num [1:50000] 568670 761732 830791 867663 916549 ...
##   .. ..- attr(*, "names")= chr [1:50000] "1:568670" "1:761732" "1:830791" "1:867663" ...
##  $ chr     :List of 1
##   ..$ : Named chr [1:50000] "1" "1" "1" "1" ...
##   .. ..- attr(*, "names")= chr [1:50000] "1:568670" "1:761732" "1:830791" "1:867663" ...
##  $ cpra    :List of 1
##   ..$ : Named chr [1:50000] "1_568670_C_T" "1_761732_C_T" "1_830791_T_C" "1_867663_T_C" ...
##   .. ..- attr(*, "names")= chr [1:50000] "1:568670" "1:761732" "1:830791" "1:867663" ...
##  $ map     :List of 1
##   ..$ : Named num [1:50000] 0.000384 0.492847 0.620827 0.620827 0.69188 ...
##   .. ..- attr(*, "names")= chr [1:50000] "1:568670" "1:761732" "1:830791" "1:867663" ...
##  $ nmiss   :List of 1
##   ..$ : Named int [1:50000] 0 0 0 0 0 0 0 0 0 0 ...
##   .. ..- attr(*, "names")= chr [1:50000] "1:568670" "1:761732" "1:830791" "1:867663" ...
##  $ af      :List of 1
##   ..$ : Named num [1:50000] 0.103 0.355 0.264 0.495 0.475 ...
##   .. ..- attr(*, "names")= chr [1:50000] "1:568670" "1:761732" "1:830791" "1:867663" ...
##  $ af1     :List of 1
##   ..$ : Named num [1:50000] 0.103 0.355 0.264 0.495 0.475 ...
##   .. ..- attr(*, "names")= chr [1:50000] "1:568670" "1:761732" "1:830791" "1:867663" ...
##  $ af2     :List of 1
##   ..$ : Named num [1:50000] 0.897 0.645 0.736 0.505 0.525 ...
##   .. ..- attr(*, "names")= chr [1:50000] "1:568670" "1:761732" "1:830791" "1:867663" ...
##  $ maf     :List of 1
##   ..$ : Named num [1:50000] 0.103 0.355 0.264 0.495 0.475 ...
##   .. ..- attr(*, "names")= chr [1:50000] "1:568670" "1:761732" "1:830791" "1:867663" ...
##  $ hom     :List of 1
##   ..$ : Named num [1:50000] 0.812 0.539 0.61 0.496 0.498 ...
##   .. ..- attr(*, "names")= chr [1:50000] "1:568670" "1:761732" "1:830791" "1:867663" ...
##  $ het     :List of 1
##   ..$ : Named num [1:50000] 0.188 0.461 0.39 0.504 0.502 ...
##   .. ..- attr(*, "names")= chr [1:50000] "1:568670" "1:761732" "1:830791" "1:867663" ...
##  $ n0      :List of 1
##   ..$ : Named int [1:50000] 4015 2071 2704 1266 1371 2305 4983 4229 4816 2667 ...
##   .. ..- attr(*, "names")= chr [1:50000] "1:568670" "1:761732" "1:830791" "1:867663" ...
##  $ n1      :List of 1
##   ..$ : Named int [1:50000] 942 2303 1948 2518 2510 2169 17 739 180 1988 ...
##   .. ..- attr(*, "names")= chr [1:50000] "1:568670" "1:761732" "1:830791" "1:867663" ...
##  $ n2      :List of 1
##   ..$ : Named int [1:50000] 43 626 348 1216 1119 526 0 32 4 345 ...
##   .. ..- attr(*, "names")= chr [1:50000] "1:568670" "1:761732" "1:830791" "1:867663" ...
##  $ nchr    : int 1
```

Quality control of genotype data

In general it advisable to perform quality control of the genotype data. The quality control include removing markers with low genotyping rate, low minor allele frequency, not in Hardy-Weinberg Equilibrium. The function `gfilter()` can be used for filtering of markers:

```
rsids <- gfilter(Glist = Glist, excludeMAF = 0.05, excludeMISS = 0.05,
    excludeCGAT = TRUE, excludeINDEL = TRUE, excludeDUPS = TRUE, excludeHWE = 1e-12,
    excludeMHC = FALSE)
```

The ´gfilter´ function output the number of variants removed in the different quality control steps.

Session 4: Compute GWAS summary statistics

One of the first step in PRS analyses is to generate or obtain GWAS summary statistics. Ideally these will correspond to the most powerful GWAS results available on the phenotype under study. In this example, we will use GWAS on the simulated disease phenotype. We will use only a subset of the data (training data) in the GWAS and the remaining subset of the data (validation data) to assess the accuracy of the polygenic risk scores. In the example below we only compute summary statistics for the markers that fulfill the quality control criteria.

Define the response variable

```
y <- pheno[, 3]
names(y) <- pheno[, 1]
```

Create design matrix for the explanatory variables

```
X <- model.matrix(~V3 + V4 + V5 + V6 + V7 + V8 + V9 + V10 + V11 + V12 +
    V13 + V14, data = covar)
rownames(X) <- covar$V1
X <- X[names(y), ]
sum(names(y) %in% rownames(X))
```

```
## [1] 5000
```

Define training and validation samples

```
train <- sample(names(y), 4000)
valid <- names(y)[!names(y) %in% train]
```

Computation of GWAS summary statistics

The function `glma` can be used for computing GWAS summary statistics. Currently this function only fit a simple linear regression model, but we plan to add further modeling approached in a future release.

```
stat <- glma(y = y[train], X = X[train, ], Glist = Glist)
```

Explore the output (stat) form the `glma` function:

```
dim(stat)
```

```
## [1] 50000    13
```

```
head(stat)
```

```
##             rsids chr    pos a1 a2     af            b         seb       stat
## 1:568670 1:568670   1 568670  C  T 0.1028  0.001656243 0.006976217  0.2374128
## 1:761732 1:761732   1 761732  C  T 0.3555  0.013676115 0.006878712  1.9881796
## 1:830791 1:830791   1 830791  T  C 0.2644 -0.001669526 0.006855669 -0.2435248
## 1:867663 1:867663   1 867663  T  C 0.4950 -0.001830340 0.006893502 -0.2655167
## 1:916549 1:916549   1 916549  A  G 0.4748  0.002758316 0.006903852  0.3995328
## 1:956852 1:956852   1 956852  C  T 0.3221 -0.002130015 0.006830618 -0.3118335
##                   p    n       ww        wy
## 1:568670 0.81234872 3998 3881.582  6.428845
## 1:761732 0.04685991 3998 3988.517 54.547417
## 1:830791 0.80761133 3998 4019.285 -6.710299
## 1:867663 0.79062517 3998 3975.278 -7.276108
## 1:916549 0.68952198 3998 3963.278 10.931973
## 1:956852 0.75518334 3998 4048.782 -8.623967
```

Session 5: Compute sparse LD matrices

Polygenic risk scoring based on summary statistics require the construction of a reference linkage disequilibrium (LD) correlation matrix. The LD matrix corresponds to the correlation between the genotypes of genetic variants across the genome. Here we use a sparse LD matrix approach using a fixed window approach (e.g. number of markers, 1 cM or 1000kb), which sets LD correlation values outside this window to zero.

The function `gprep` can be used to compute sparse LD matrices which are stored on disk. The r^2 metric used is the pairwise correlation between markers (allele count alternative allele) in a specified region of the genome. Although this step can be slow unless R is linked to a fast BLAS it is typically only done once (or a few times).

Define filenames for the sparse LD matrices.

```
ldfiles <- "C:\\Users\\au223366\\Dropbox\\Projects\\Summer_course\\human.ld"
```

Compute sparse LD using only the filtered rsids

```
Glist <- gprep(Glist, task = "sparseld", msize = 1000, rsids = rsids, ldfiles = ldfiles,
    overwrite = TRUE)
saveRDS(Glist, file = "C:\\Users\\au223366\\Dropbox\\Projects\\Summer_course\\Glist_sparseLD_1k.RDS",
    compress = FALSE)
```

Session 6: Compute PRS using clumping and thresholding (C+T)

Polygenic risk scoring using clumping and thresholding is a relative simple and robust method. Linkage disequilibrium makes identifying the contribution from causal independent genetic variants extremely challenging. One way of approximately capturing the right level of causal signal is to perform clumping, which removes markers in ways that only weakly correlated SNPs are retained but preferentially retaining the SNPs most associated with the phenotype under study. The clumping procedure uses a statistic (usually P-value) to sort the markers by importance (e.g. keeping the most significant ones). It takes the first one (e.g. most significant marker) and removes markers (i.e. set their effect to zero) if they are too correlated (e.g. $r^2 > 0.9$) with this one in a window around it. As opposed to pruning, this procedure makes sure that this marker is never removed, keeping at least one representative marker by region of the genome. Then it goes on with the next most significant marker that has not been removed yet.

Clumping and thresholding

Clumping can be performed using the `adjStat()`-function in `qgg`. The input to the function is the summary statistic (`stat`), information about sparse LD matrices which is in the `Glist`, a threshold of linkage disequilibrium (e.g. $r^2 = 0.9$) and thresholds for P-values (`threshold = c(0.001, 0.05, ...)`):

```
threshold <- c(1e-05, 1e-04, 0.001, 0.005, 0.01, 0.05, 0.1, 0.2, 0.5, 1)
statAdj <- adjStat(Glist = Glist, stat = stat, r2 = 0.9, threshold = threshold)
```

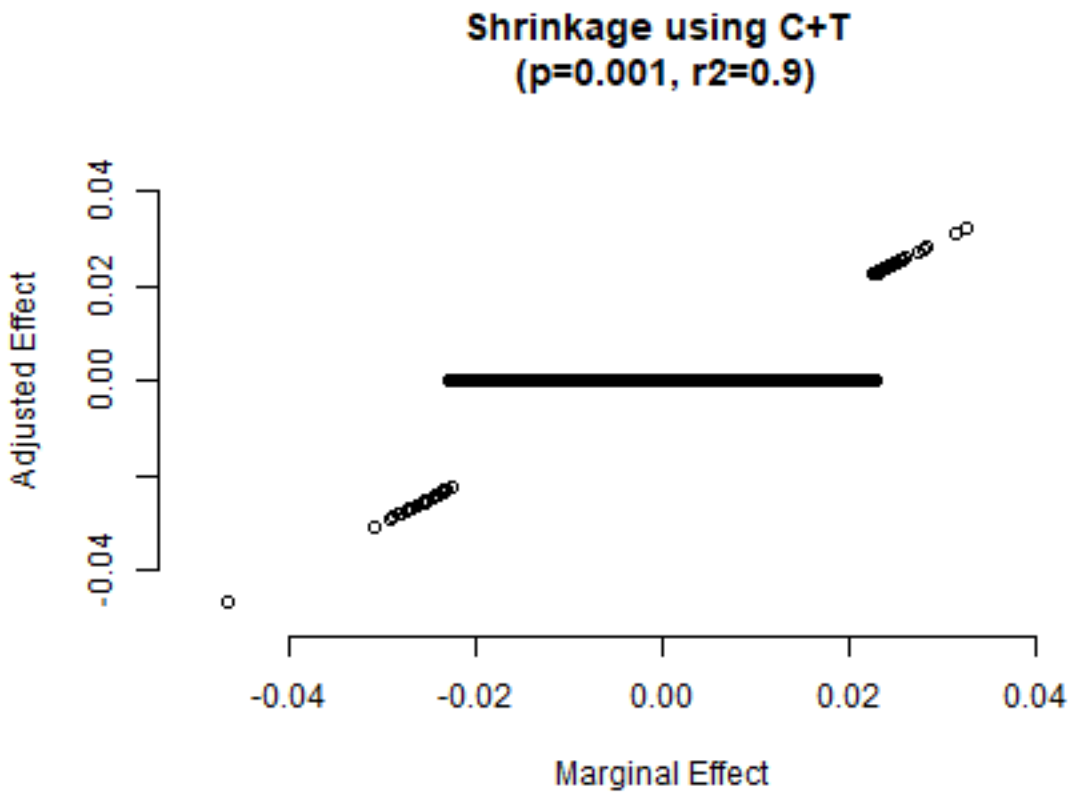
Explore the output (statAdj) using the `head`function:

```
head(statAdj)
```

```
##             rsids chr    pos a1 a2     af            b b_1e.05 b_1e.04 b_0.001
## 1:568670 1:568670   1 568670  C  T 0.1028  0.001656243       0       0       0
## 1:761732 1:761732   1 761732  C  T 0.3555  0.013676115       0       0       0
## 1:830791 1:830791   1 830791  T  C 0.2644 -0.001669526       0       0       0
## 1:867663 1:867663   1 867663  T  C 0.4950 -0.001830340       0       0       0
## 1:916549 1:916549   1 916549  A  G 0.4748  0.002758316       0       0       0
## 1:956852 1:956852   1 956852  C  T 0.3221 -0.002130015       0       0       0
##          b_0.005 b_0.01     b_0.05      b_0.1      b_0.2      b_0.5
## 1:568670       0      0 0.00000000 0.00000000 0.00000000 0.00000000
## 1:761732       0      0 0.01367612 0.01367612 0.01367612 0.01367612
## 1:830791       0      0 0.00000000 0.00000000 0.00000000 0.00000000
## 1:867663       0      0 0.00000000 0.00000000 0.00000000 0.00000000
## 1:916549       0      0 0.00000000 0.00000000 0.00000000 0.00000000
## 1:956852       0      0 0.00000000 0.00000000 0.00000000 0.00000000
##                   b_1
## 1:568670  0.001656243
## 1:761732  0.013676115
## 1:830791 -0.001669526
## 1:867663 -0.001830340
## 1:916549  0.002758316
## 1:956852 -0.002130015
```

A plot of the un-adjusted marker effect (from the `stat` data frame) against the adjusted marker effects (from the the `statAdj` data frame) illustrates that the C+T procedure keep only the most significant marker effects and is setting a large number of marker effects to zero (i.e. remove their effect).

```
plot(y = statAdj[rownames(stat), "b_0.001"], col = 1, x = stat$b, xlab = "Marginal Effect",
    ylab = "Adjusted Effect", frame.plot = FALSE, ylim = c(-0.05, 0.05),
    xlim = c(-0.05, 0.05), main = "Shrinkage using C+T \n (p=0.001, r2=0.9)")
```



Compute polygenic risk scores

For each of the P-value thresholds chosen in the C+T procedure a PRS is computed as:

$$PRS = \sum_{i=1}^{m} X_i \hat{b}_i$$

where X_i is the genotype vector, and $\hat{b}_i$ the weight of the i'th single genetic marker. The PRS are computed using the `gscore()` function. The input to the function is the adjusted summary statistic (`adjStat`), and information about the genotypes which are in the `Glist`:

```
prs <- gscore(Glist = Glist, stat = statAdj)
```

Explore polygenic scores

It is always important to explore the PRS computed.

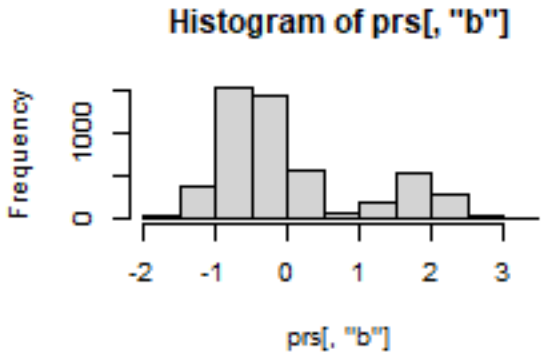
```
head(prs)
```

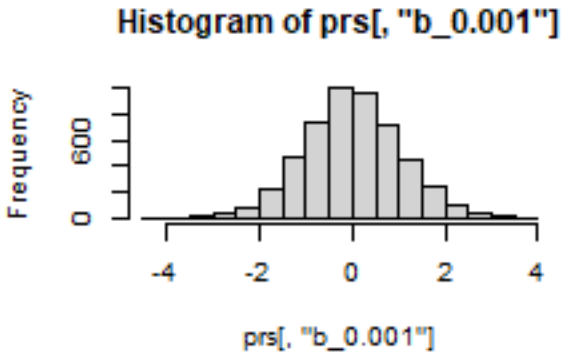
```
##                   b    b_1e.05    b_1e.04     b_0.001    b_0.005     b_0.01
## IND1000 -0.93363124 -0.0628105 -0.4344330 -0.02459312  0.5236039  0.1707728
## IND1001  0.03202941 -0.7188466 -0.2649802 -0.48312749  0.9634211  0.8657329
## IND1002 -0.73027197 -0.7595138 -1.1394759 -2.25664885 -1.7346148 -2.5484331
## IND1003  2.35733509 -0.7188466  0.9843286  1.17791465  0.6326977  0.8744531
## IND1004  2.23849601  0.9754915  0.8478963  1.06539177  1.8165957  0.8942499
## IND1005 -0.39245501 -1.4765113 -1.1718997 -0.40228597 -0.3905964 -0.7875224
##              b_0.05       b_0.1      b_0.2       b_0.5         b_1
## IND1000 -0.50668314 -0.638924595 -0.5955633 -0.93427262 -0.93363124
## IND1001  1.07053438  0.554238053  0.3139273  0.04683049  0.03202941
## IND1002 -1.40010432 -0.671594726 -0.7199750 -0.60260853 -0.73027197
## IND1003  2.04651589  1.996397794  2.4891906  2.37177555  2.35733509
## IND1004  2.34674669  2.252387657  2.3673747  2.18043862  2.23849601
## IND1005 -0.03500853  0.001646114 -0.5432879 -0.37918881 -0.39245501
```

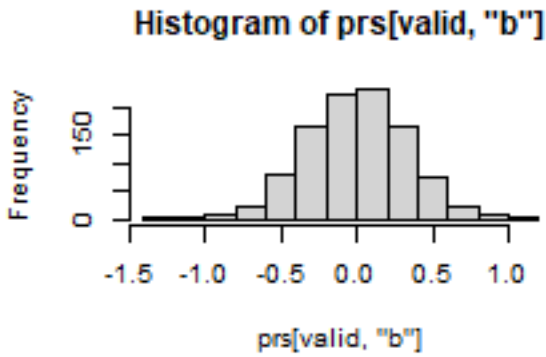
```
cor(prs)
```

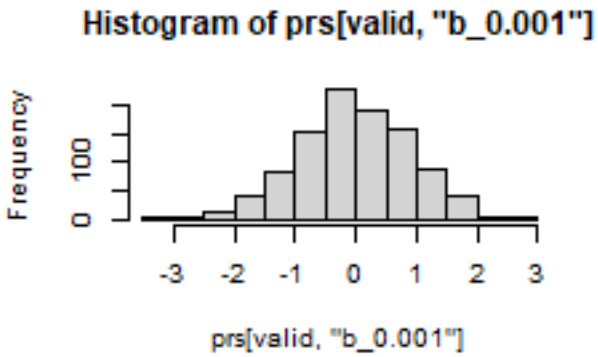
```
##                 b   b_1e.05   b_1e.04   b_0.001   b_0.005    b_0.01    b_0.05
## b       1.0000000 0.1492487 0.2474547 0.4206682 0.6033828 0.6984484 0.8861401
## b_1e.05 0.1492487 1.0000000 0.6012332 0.3612355 0.2570072 0.2171601 0.1633672
## b_1e.04 0.2474547 0.6012332 1.0000000 0.6200611 0.4347663 0.3698081 0.2874852
## b_0.001 0.4206682 0.3612355 0.6200611 1.0000000 0.7063663 0.6089880 0.4797717
## b_0.005 0.6033828 0.2570072 0.4347663 0.7063663 1.0000000 0.8650944 0.6776496
## b_0.01  0.6984484 0.2171601 0.3698081 0.6089880 0.8650944 1.0000000 0.7844649
## b_0.05  0.8861401 0.1633672 0.2874852 0.4797717 0.6776496 0.7844649 1.0000000
## b_0.1   0.9364655 0.1564770 0.2692708 0.4508634 0.6395761 0.7434395 0.9480078
## b_0.2   0.9716919 0.1502060 0.2577306 0.4363324 0.6219938 0.7209834 0.9138898
## b_0.5   0.9957755 0.1496925 0.2484542 0.4218900 0.6056616 0.7008632 0.8898793
## b_1     1.0000000 0.1492487 0.2474547 0.4206682 0.6033828 0.6984484 0.8861401
##             b_0.1     b_0.2     b_0.5       b_1
## b       0.9364655 0.9716919 0.9957755 1.0000000
## b_1e.05 0.1564770 0.1502060 0.1496925 0.1492487
## b_1e.04 0.2692708 0.2577306 0.2484542 0.2474547
## b_0.001 0.4508634 0.4363324 0.4218900 0.4206682
## b_0.005 0.6395761 0.6219938 0.6056616 0.6033828
## b_0.01  0.7434395 0.7209834 0.7008632 0.6984484
## b_0.05  0.9480078 0.9138898 0.8898793 0.8861401
## b_0.1   1.0000000 0.9642190 0.9411723 0.9364655
## b_0.2   0.9642190 1.0000000 0.9763079 0.9716919
## b_0.5   0.9411723 0.9763079 1.0000000 0.9957755
## b_1     0.9364655 0.9716919 0.9957755 1.0000000
```

```
layout(matrix(1:4, ncol = 2, byrow = TRUE))
hist(prs[, "b"])
hist(prs[, "b_0.001"])
hist(prs[valid, "b"])
hist(prs[valid, "b_0.001"])
```









Evalute polygenic scores

The *P*-value threshold that provides the "best-fit" PRS under the C+T method is usually unknown. To approximate the "best-fit" PRS, we can perform a regression between PRS calculated at a range of *P*-value thresholds and then select the PRS that explains the highest proportion of phenotypic variance (e.g. R2) or has the highest AUC. This can be achieved using `acc()`-function as follows:

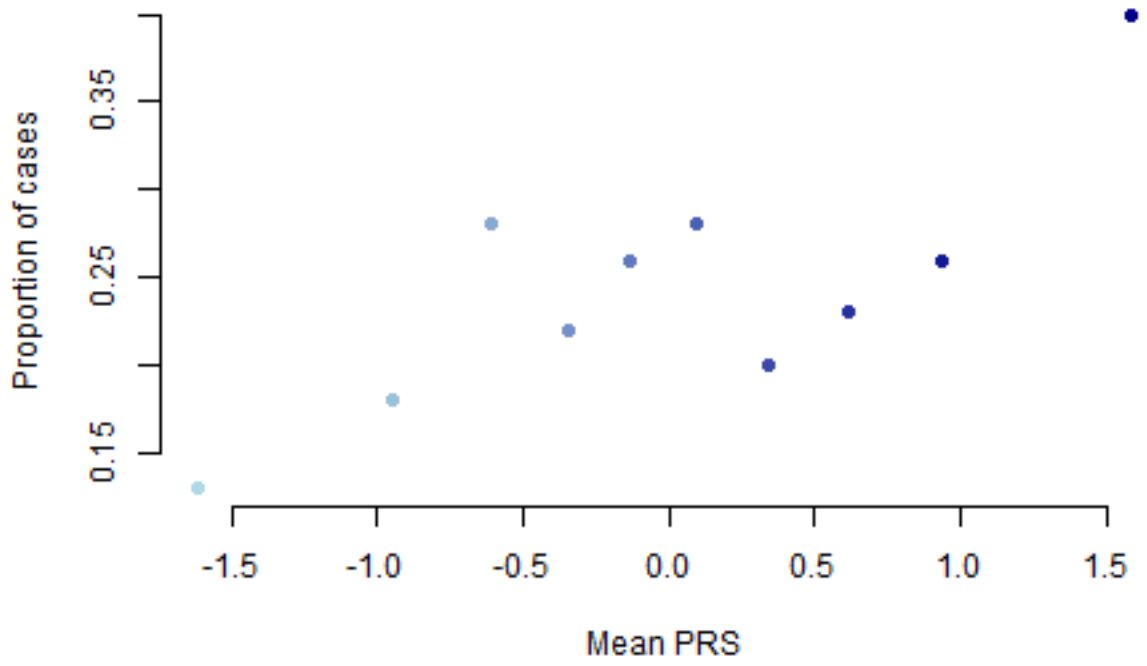
```
paCT <- acc(yobs = y[valid], ypred = prs[valid, ], typeoftrait = "binary")
paCT
```

```
##           Corr    R2 Nagel R2   AUC intercept slope  MSPE
## b       0.019 0.000    0.001 0.507     0.244 0.025 0.351
## b_1e.05 0.072 0.005    0.008 0.542     0.245 0.031 1.179
## b_1e.04 0.096 0.009    0.014 0.552     0.243 0.044 1.057
## b_0.001 0.128 0.016    0.024 0.575     0.245 0.060 0.976
## b_0.005 0.118 0.014    0.021 0.575     0.242 0.063 0.792
## b_0.01  0.072 0.005    0.008 0.542     0.244 0.042 0.741
## b_0.05  0.038 0.001    0.002 0.522     0.244 0.030 0.522
## b_0.1   0.028 0.001    0.001 0.509     0.244 0.026 0.450
## b_0.2   0.031 0.001    0.001 0.514     0.244 0.035 0.390
## b_0.5   0.013 0.000    0.000 0.502     0.244 0.017 0.357
## b_1     0.019 0.000    0.001 0.507     0.244 0.025 0.351
```

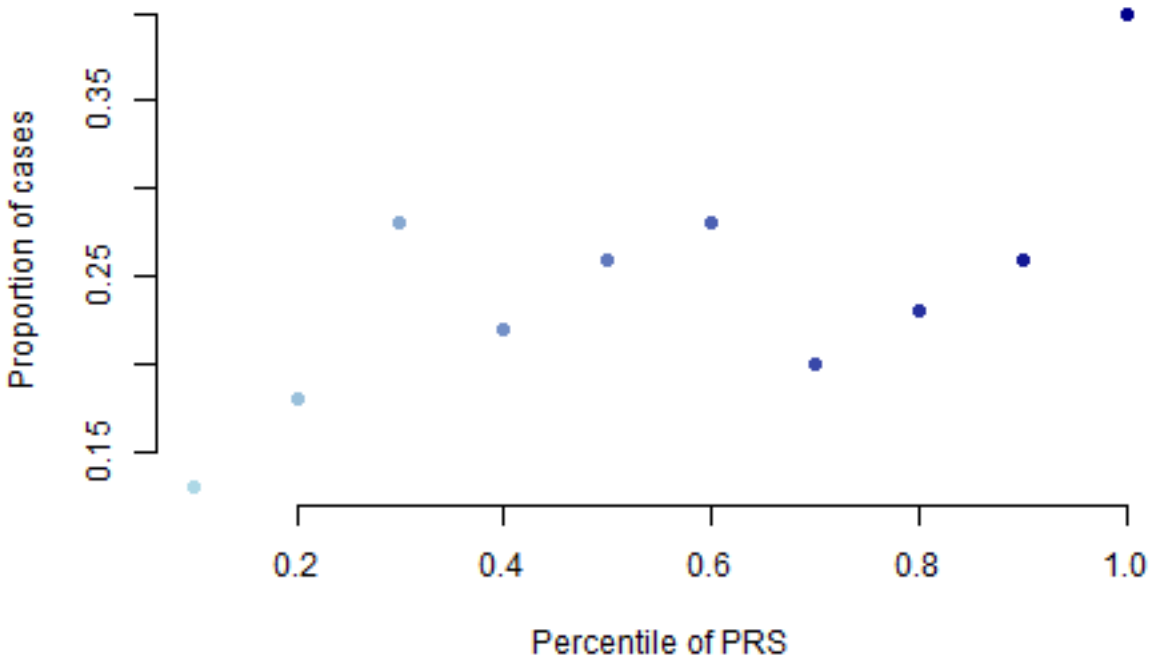
Plot polygenic scores

For visualization, the PRS can be divided into groups (e.g., deciles), and the disease prevalence within each group was computed.

```
yobs <- y[valid]
ypred <- prs[names(y[valid]), which.max(paCT[, "AUC"])]

nbin <- 10
qsets <- qgg:::splitWithOverlap(names(ypred)[order(ypred)], length(ypred)/nbin,
    0)
qy <- sapply(qsets, function(x) {
    mean(yobs[x])
})
qg <- sapply(qsets, function(x) {
    mean(ypred[x])
})

colfunc <- colorRampPalette(c("lightblue", "darkblue"))

plot(y = qy, x = qg, pch = 19, ylab = "Proportion of cases", xlab = "Mean PRS",
    col = colfunc(nbin), frame.plot = FALSE)
```




```
plot(y = qy, x = (1:nbin)/nbin, pch = 19, ylab = "Proportion of cases",
    xlab = "Percentile of PRS", col = colfunc(nbin), frame.plot = FALSE)
```


Proportion of cases
0.15
0.25
0.35
0.2
0.4
0.6
0.8
1.0
Percentile of PRS

Session 7: Compute PRS using different Bayesian Linear Regression (BLR) models

Bayesian linear regression models have been proposed as a unified framework for gene mapping, genomic risk scoring, estimation of genetic parameters and effect size distribution. BLR methods use an iterative algorithm for estimating joint marker effects that account for LD and allow for differential shrinkage of marker effects. Estimation of the joint marker effects depend on additional model parameters such as a probability of being causal (π), an overall marker variance (σ_b^2), and residual variance (σ_e^2). Estimation of model parameters can be done using MCMC techniques by sampling from fully conditional posterior distributions.

Genomic risk scoring using Bayesian Linear Regression (BLR) models is a very flexible approach for accounting for the underlying genetic architecture of the traits. It can be implemented using GWAS summary statistics and a reference linkage disequilibrium (LD) correlation matrix. Ideally the summary statistics will correspond to the most powerful GWAS results available on the phenotype under study. In this example, we will use the summary statistics output the ´glma´ function that fit a linear regression model on the quantitative trait. We will use only a subset of the data (training data) in the GWAS and the remaining subset of the data (validation data) to assess the accuracy of the genomic risk scores.

Different BLR models can be fitted in the `qgg`-package in the function `gbayes()`, where the argument `method=` specifies which prior marker variance should be used. BLR models can be fitted using individual level genotype and phenotype data or based on GWAS summary statistics and a reference linkage disequilibrium (LD) correlation matrix.

Fit BLR model using a Bayes N prior for the marker variance

In the Bayes N approach the prior the marker effect, b, follows a priori a normal distribution with a marker effect variance which is constant across markers:

```
fitN <- gbayes(stat = stat, Glist = Glist, method = "bayesN", nit = 1000)
grsN <- gscore(Glist = Glist, stat = fitN$stat)
paN <- acc(yobs = y[valid], ypred = grsN[valid, ], typeoftrait = "binary")
paN
```

```
##      Corr R2 Nagel R2   AUC intercept slope  MSPE
## res 0.018  0        0 0.503     0.244 0.023 0.352
```

Fit BLR model using a Bayes A prior for the marker variance

In the Bayes A approach it is assumed that a priori we have some information on the marker variance. For instance, this can be σ_b^2. Thus, we may attach some importance to this value and use it as prior information for

```
fitA <- gbayes(stat = stat, Glist = Glist, method = "bayesA", nit = 1000)
grsA <- gscore(Glist = Glist, stat = fitA$stat)
paA <- acc(yobs = y[valid], ypred = grsA[valid, ], typeoftrait = "binary")
paA
```

```
##      Corr    R2 Nagel R2   AUC intercept slope  MSPE
## res 0.062 0.004    0.006 0.533     0.244 0.072 0.366
```

Fit BLR model using a Bayes C prior for the marker variance

In the Bayes C approach the marker effects, b, are a priori assumed to be sampled from a mixture with a point mass at zero and univariate normal distribution conditional on a common marker effect variance:

```
fitC <- gbayes(stat = stat, Glist = Glist, method = "bayesC", nit = 1000)
grsC <- gscore(Glist = Glist, stat = fitC$stat)
paC <- acc(yobs = y[valid], ypred = grsC[valid, ], typeoftrait = "binary")
paC
```

```
##      Corr    R2 Nagel R2   AUC intercept slope  MSPE
## res 0.123 0.015    0.023 0.568     0.244 0.062 0.894
```

Fit BLR model using a Bayes R prior for the marker variance

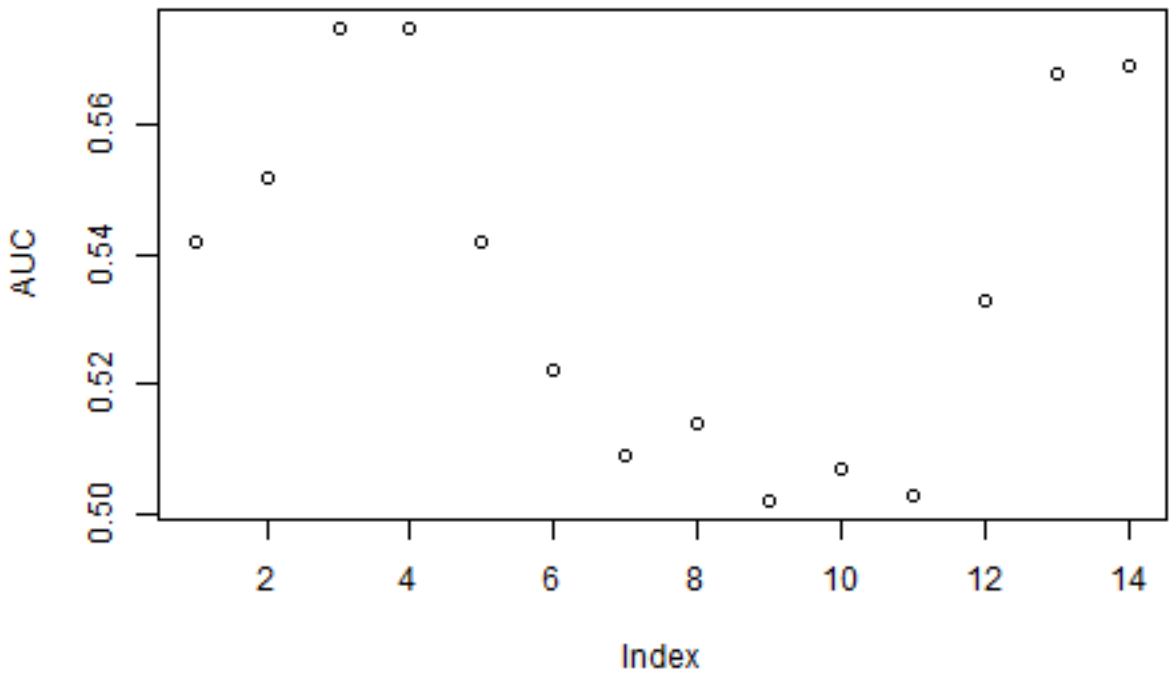
In the Bayes R approach the marker effects, b, are a priori assumed to be sampled from a mixture with a point mass at zero and univariate normal distributions conditional on a common marker effect variance:

```
fitR <- gbayes(stat = stat, Glist = Glist, method = "bayesR", nit = 1000)
grsR <- gscore(Glist = Glist, stat = fitR$stat)
paR <- acc(yobs = y[valid], ypred = grsR[valid, ], typeoftrait = "binary")
paR
```

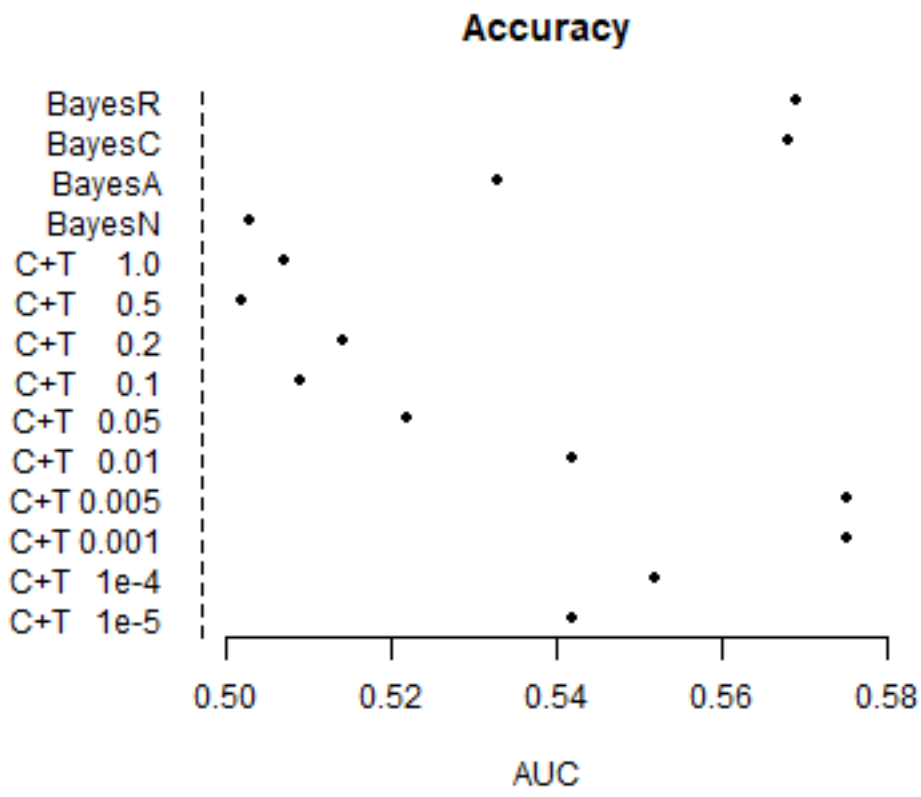
```
##      Corr    R2 Nagel R2   AUC intercept slope  MSPE
## res 0.119 0.014    0.021 0.569     0.244  0.09 0.511
```

Plot comparing AUC for the different BLR models

```
auc <- c(paCT[-1, "AUC"], paN[, "AUC"], paA[, "AUC"], paC[, "AUC"], paR[,
    "AUC"])
plot(auc, ylab = "AUC")
```

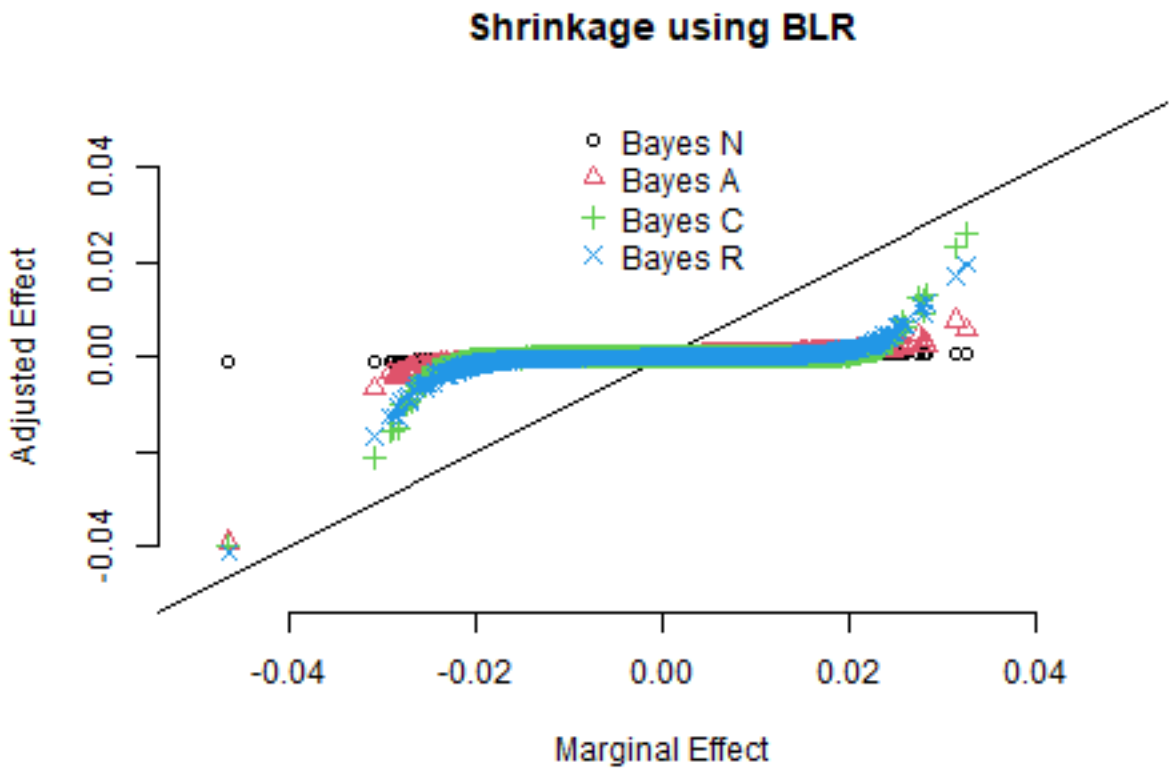



```
names(auc) <- c("C+T   1e-5", "C+T   1e-4", "C+T 0.001", "C+T 0.005", "C+T   0.01",
    "C+T   0.05", "C+T     0.1", "C+T     0.2", "C+T     0.5", "C+T     1.0",
    "BayesN", "BayesA", "BayesC", "BayesR")
qgg:::plotForest(x = auc, sd = rep(0, length(auc)), reorder = FALSE, xlab = "AUC",
    main = "Accuracy")
```



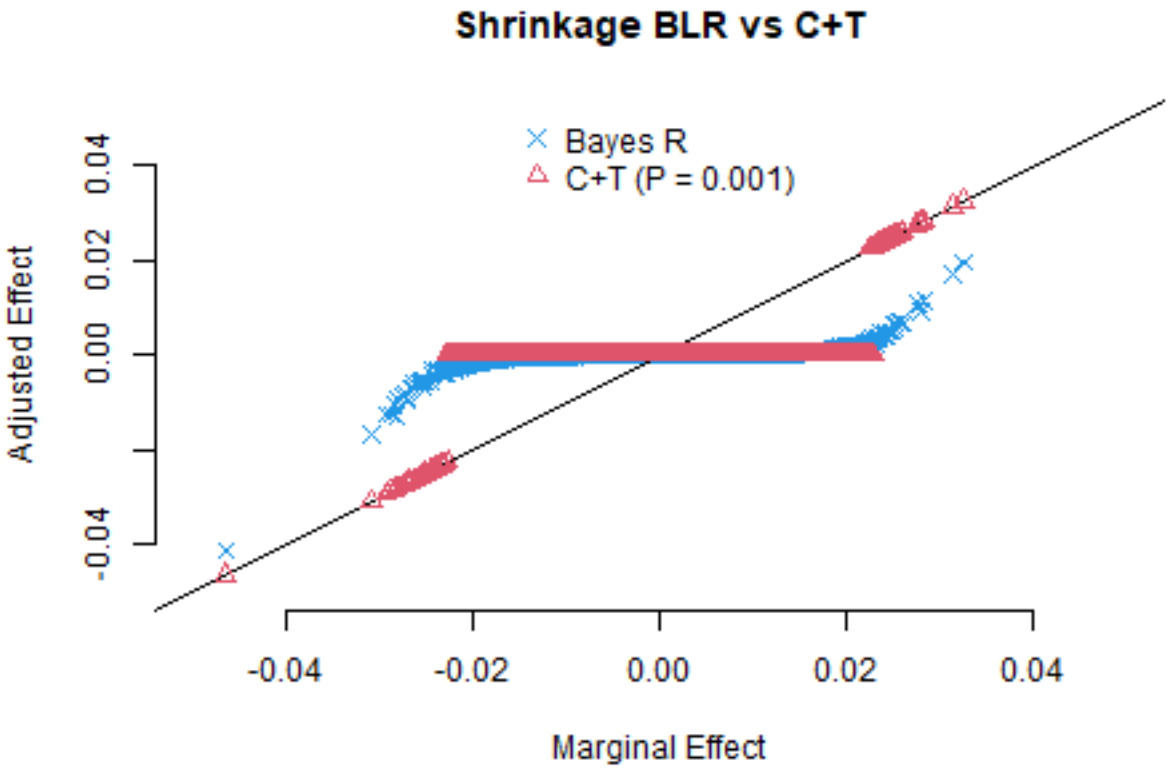
Plot comparing degree of shrinkage and for the different BLR models

```
plot(y = fitN$stat[rownames(stat), "bm"], col = 1, pch = 1, x = stat$b,
    xlab = "Marginal Effect", ylab = "Adjusted Effect", frame.plot = FALSE,
    ylim = c(-0.05, 0.05), xlim = c(-0.05, 0.05), main = "Shrinkage using BLR")
abline(a = 0, b = 1)
points(y = fitA$stat[rownames(stat), "bm"], x = stat$b, col = 2, pch = 2)
points(y = fitC$stat[rownames(stat), "bm"], x = stat$b, col = 3, pch = 3)
points(y = fitR$stat[rownames(stat), "bm"], x = stat$b, col = 4, pch = 4)
legend("top", legend = c("Bayes N", "Bayes A", "Bayes C", "Bayes R"), col = 1:4,
    pch = 1:4, bty = "n", text.col = "black", horiz = F)
```



Plot comparing degree of shrinkage for the BayesR BLR model and clumping and thresholding

```
plot(y = fitR$stat[rownames(stat), "bm"], col = 4, pch = 4, x = stat$b,
    xlab = "Marginal Effect", ylab = "Adjusted Effect", frame.plot = FALSE,
    ylim = c(-0.05, 0.05), xlim = c(-0.05, 0.05), main = "Shrinkage BLR vs C+T")
abline(a = 0, b = 1)
points(y = statAdj[rownames(stat), "b_0.001"], x = stat$b, col = 2, pch = 2)
legend("top", legend = c("Bayes R", "C+T (P = 0.001)"), col = c(4, 2),
    pch = c(4, 2), bty = "n", text.col = "black", horiz = F)
```


Shrinkage BLR vs C+T
Bayes R
C+T (P = 0.001)
Adjusted Effect
-0.04
0.00
0.02
0.04
-0.04
-0.02
0.00
0.02
0.04
Marginal Effect