GWAS Analysis Practical

Wellcome Trust Advanced Course – June 2015 – Durban, SA

1: Introduction

The series of practicals today will introduce you to analyzing Genome Wide Association Study (GWAS) datasets using a program called PLINK, which is a freely available GWAS analysis toolkit. PLINK has many functions including those related to data organization, formatting, quality control, association testing, population stratification and much more. Details about PLINK and its documentation are available at the reference section at the end of these practicals.

Open a directory browser and navigate to the *data/GEIA/Practicals/07_GWAS_analysis* folder.
Click on the Date Modified header to sort the files by date (newest first). This will allow you to see your results files appear and make it easier for you to open them by double-clicking.

We will use the `chr19-example.vcf/sample/fam` and `snp-example.ped/map/sample/fam` files in the *GWAS_analysis* folder. These files have been simulated and do not represent any real data, but instead serve to illustrate important points in working with GWAS data. We will use PLINK commands in the terminal to perform analyses. We can use the text editor, spreadsheet program and R (as introduced earlier this week) to help interpret results. Checkpoint questions in the practicals are **shown in bold**.

Open a terminal and navigate to the practical directory now (ask if you do not know where it is.)

```
      cd /media/ubuntu/data/GEIA/Practicals/07_GWAS_analysis
```

type `ls` to view the file list.

PLINK is run from the terminal command line by typing “plink”. In these practicals we will show commands to be typed in the terminal like this:

```
plink --version
```

Here we have simply asked PLINK what its software version is, using the command "`--version`". It will tell you its version, and when it was last updated.

We can also run a more complicated command, like this:

```
plink --freq  --file snp-example
```

The "`--freq`" command asks PLINK to calculate allele frequencies. "`--file snp-example`" tells PLINK to read in files that start with "snp-example". The two files it reads in are *snp-example.ped* and *snp-example.map*. The *.map* file which contains information about the SNPs included and the *.ped* file which contains information about the individuals and their genotypes, with one line per individual.

PLINK has written two output files in the directory we are working in, 1. a log file called *plink.log* that summarizes what it has done (this is the same as the output that appeared in the terminal window), and 2. *plink.freq*, which contains the allele frequency information. You can look at all input and output files in your text editor.

Q. How many SNPs were in this dataset? What are their names? What are their allele frequencies?

Q. How many samples are included in total?

We can do something more interesting with these files by running:

```
plink --file snp-example --assoc
```

Now PLINK has generated two files in the directory we are working in: *plink.log* and *plink.assoc*. The log file simply captures the status information that PLINK reports with each run.

Q. How many cases and controls are included in this data?

If we look in the assoc file we will see that this SNP, rs8135996, is associated with our phenotype.

Q. What is the p-value of association? What is the odds ratio?

You can find more about the format of this file here:

https://www.cog-genomics.org/plink2/formats#assoc.

Q. Look at the frequencies of the allele in cases and controls. Is the minor allele protective, or does it confer a risk of being a case in this data?

The final step in this introduction is to learn to rename PLINK's output files, since we'll be generating lots of them in the practicals.

```
plink --file snp-example --assoc --out getting-started
```

Now the output files will be named *getting-started.log* and *getting-started.assoc*. This is the basic pattern to working with PLINK: specifying input files and analyses, along with an output name to save results.

Reading files in a different format

When we analyze real data, we won't always have the files ready in ".ped" and ".map" format. Instead, you might have other formats, such as the VCF format that we mentioned in the Public Resources talk. PLINK can read in many of these file formats, but you need to tell it more about the files it is reading in.

In the working directory you can see two files*, snp-example.vcf* and *snp-example.samples*. These contain the same data as the snp-example.ped and snp-example.map files, but in a different format. If you look at the *snp-example.samples* in a text editor, you will see a list of sample names ("sample1", "sample2", etc), followed by a few numbers. The first number is the case-control status of the sample (1 means unaffected, 2 means affected), the second is the the sex of the sample (1 means male, 2 means female). The other file, *snp-example.vcf*, is in the VCF format that we saw in the Public Resources lecture, and contains the genotypes for each sample. You can look at the file in a text editor, but you might find it hard to read.

We can use PLINK to convert these files to .ped and .map files:

```
plink --recode --vcf snp-example.vcf --pheno snp-example.samples --
update-sex snp-example.samples 2 --out snp-example-recode
```

Here we are using the command "`--recode`" to rewrite the data into a new format. "`--vcf snp-example.vcf`" tells it to read the genotypes from the VCF file, "`--pheno snp-example.samples`" tells Plink to take the phenotypes from the sample file, and "`--update-sex snp-example.sex 2`" tells it to take gender information from the second column of the sample. Finally, "`--out snp-example-recode`" tells PLINK the prefix to use when writing the new files.

You can see that we have now made snp-example-recode.ped and snp-exampe-recode.map files that are in the format PLINK expects.

1a: Quality Control Practical

We will now work with a set of data files containing many SNPs from chromosome 19 genotyped on controls and cases. Data from a GWAS would contain SNPs at this density across the entire genome, but we will focus on just one chromosome to make the exercises more tractable.

The key files are:
chr19-example.vcf
chr19-example.samples

As we saw above, we can convert these to files PLINK can read. Here, instead of converting to .ped files with “--recode", we will convert them to so-called “binary PLINK” or .bed format using “--make-bed”:

```
plink --make-bed --vcf chr19-example.vcf --pheno chr19-
example.samples --update-sex chr19-example.samples  2 --out
chr19-example --keep-allele-order
```

Like the *.ped* and *.map* above, these files contain information about the samples and SNPs, as well as the genotypes for each of the samples at each of the SNPs. Unlike the human-readable text *.ped* file we used before, these data are in “binary ped” format (*.bed*). This format is a compressed format which saves space and speeds up analysis. Information on samples can be found in the *.fam* file and information on SNPs in the *.bim* file. We've used the --keep-allele-order option here to make sure plink preserves the two alleles at each SNP in the same order as in the VCF file (we'll see why this is important later)

We can load data in the binary plink format using the `--bfile` option. For instance, to calculate allele frequencies we use:

```
plink --bfile chr19-example --freq
```

Q. How many SNPs and samples are in this dataset?

Reformatting between .bed/.bim/.fam and .ped/.map is easy

```
plink --bfile chr19-example --recode --out chr19-example
```

Note that this may take a few moments to run. Look at the file sizes in the directory (`ls -l [minus ell]` on the terminal prompt).

Q. What are the relative sizes of *.ped, .bed* and *.vcf* files?

Note: Many other formats are in use for genetic data, including 'GEN' format, 'BGEN' format, and BCF (binary VCF) format.

There are many different QC metrics that we can calculate for our dataset. These metrics can tell us about the quality of loci (i.e. SNPs), and of samples. For instance, we can calculation information about missingness:

```
plink --bfile chr19-example --missing --out miss-info
```

This will produce a *.imiss* file with information about individuals and *.lmiss* with information about loci (SNPs). You can load this output into a spreadsheet program to look at it in more detail: In the directory browser right-click *miss-info.lmiss file.* Select open in Libreoffice Calc or choose other application to select Libreoffice Calc. An options window will open and in the Separator options you need to ensure that only 'separated by space' and the 'merge delimiters' boxes are checked before proceeding!

Q. Which SNP has the highest missing rate?

Another QC metric is sample heterozygosity:

```
plink --bfile chr19-example --het --out het-info
```

We can use R to visualize these results. Launch R as in previous practicals. Commands to be typed in R (rather than in the terminal) will be shaded like this:

```
In RStudio:

setwd('/media/ubuntu/data/GEIA/Practicals/07_GWAS_analysis/')

HetData<-read.table("het-info.het",h=T)

hist(HetData$F,breaks=100)
```

The graph you see shows the distribution of heterozygosity across samples. QC plots often look like this; there is a large peak of samples that lie within a range of normal values (the normal samples), and then a small number of outlier samples that are usually poor quality samples. In this case you can see that there is a large peak in heterozygosity around 0.1, with a number of outliers below 0 or above 0.2.

You can save a picture of this analysis for later reference:

```
In RStudio:

png("HetHist.png")

hist(HetData$F,breaks=100)

dev.off()
```

Next, lets plot the missingness values.

```
In RStudio:

MissData<-read.table("miss-info.imiss",h=T)

hist(MissData$F_MISS,breaks=100)

png("MissHist.png")

hist(MissData$F_MISS,breaks=100)

dev.off()
```

Again, most of the samples have low missingness (close to zero), with a number of outliers above 0.02. We can now combine these two QC metrics (missingness and heterozygosity) to select outlying samples:

```
In RStudio:

qcFails <- MissData[MissData$F_MISS > 0.02 | HetData$F > 0.2 |

HetData$F < 0,c(1,2)]

write.table(qcFails,quote=F,row.names=F,col.names=F,file="qcFai

ls.txt")
```

We can combine this list with other quality control metrics (described in the lectures) to create a clean dataset (note that although we need to break long commands in this document over two lines, all options should be typed on one continuous line in the terminal without breaks):

```
plink --bfile chr19-example --remove qcFails.txt --hwe 1e-4
include-nonctrl --geno 0.01 --maf 0.01 --make-bed --out chr19-
clean --keep-allele-order
```

Q. What filters have we applied here? (the PLINK website can help explain these options)

2: Association Practical

Now we will test for association between the SNPs in our dataset and malaria. Basic association tests can be done as follows:

```
plink --bfile chr19-example --assoc --out basic-test --keep-allele-
order
```

Load these results in the spreadsheet program to investigate them further.

Q. How many SNPs are associated? Is this what you expect?

We can again use R to visualize these data. First make sure you are in the correct practical folder:

```
In RStudio:

setwd('/media/ubuntu/data/GEIA/Practicals/07_GWAS_analysis/')
```

Now let's load the data and plot it:

```
data<-read.table("basic-test.assoc",h=T)

plot(data$BP,-log(data$P)/log(10),ylim=c(0,15))

png("basic-association.png")

plot(data$BP,-log(data$P)/log(10),ylim=c(0,15))

dev.off()
```

What does this plot of association p-values across our data tell you?

We can now test for association within the cleaned dataset we created earlier:

```
plink --bfile chr19-clean --assoc --out clean-test --keep-allele-order
```

Read this new dataset into R (as above) and look at the plot of association p-values.

```
In RStudio:

data<-read.table("clean-test.assoc",h=T)

plot(data$BP,-log(data$P)/log(10),ylim=c(0,15))

png("clean-association.png")

plot(data$BP,-log(data$P)/log(10),ylim=c(0,15))

dev.off()
```

How has cleaning the data affected our signals of association? What does that imply about the associations seen in the previous analysis?

In addition to using data cleaning to remove strong false positive associations, we are also interested in the overall distribution of test statistics. One way to look at this is to compare the observed data to our expectation under the null (remember to type each command as one long entry):

```
In RStudio:
median(data$P)

expected <- -log10( seq(0,1-1/length(data$P), by=1/length(data$P)))

plot(expected,-log10(sort(data$P)))
abline(0,1)
png("clean-qq.png")
plot(expected,-log10(sort(data$P)))
abline(0,1)
dev.off()
```

This figure is called a Q-Q plot and can be very useful in evaluating GWAS data for systematic bias.

The expected median p-value is 0.5. The diagonal line shows where the points should fall if the null hypothesis that there are no association signals were true at most SNPs.

Q. Given this information, what can you infer about our current association analysis?

Q. Using the PLINK website, can you identify how to run more general tests of association (beyond the basic test we’ve done already)?

3: Population Structure Practical

We have seen how applying appropriate quality control filters to our data eliminated many false positives, but a systematic inflation remained. The most common source of test statistic inflation is population stratification, as you have already been taught. We can use principal component analysis to measure the genetic structure within the dataset, and use these estimated components as covariates in the association analysis. You have already learned about PCA in your previous practical, and have already created PCs for these samples.

The file chr19-example.pca contains the principal components for this dataset (calculated in the previous practical), and the following command will carry out an association analysis controlling for these:

```
plink --bfile chr19-clean --logistic --covar chr19-example.pca --out
pccorrected-test --keep-allele-order --ci 0.95
```

We have now produced an analysis corrected for population structure.   Congratulations!

Note: On Thursday we will use the output from the above in a meta-analysis with data from other studies of the same phenotype. To help with this, we've used a couple of options above that will make the output consistent across studies. Firstly, as above we've used the `--keep-allele-order` option to make plink output odds ratios for the non-reference allele. (By default it will output test statistics for the minor, i.e. less common allele, but that's problematic as it might vary between studies.) We've also used the `--ci` option to tell plink to output a *standard error* along with the odds ratio. We'll use these standard errors in the meta-analysis practical tomorrow.

For now, let's use the text editor, spreadsheet and R (as in yesterday's practicals) to examine these results.

Note that the output file “`pccorrected-test.assoc.logistic`” contains information for the SNPs and also all the principal components. Make sure you extract just the SNP lines (those that have “ADD” in the TEST column):

```
In RStudio:

data<-read.table("pccorrected-test.assoc.logistic",h=T)

data <- data[data$TEST == "ADD",]
```

Q. Has stratifying by disease group corrected our inflation?

Q. Are there any disease associations in this dataset?

Q. How are the QC statistics for any associated SNPs? Do you believe these associations?

Q. Can you find out anything about the function of these associated SNPs, using the tools you learned about in the Public Resources practical?

Q. What could explain differences in association p-values in the different analyses we’ve done?

Q. Produce Q-Q plots, genome-wide p-value plots and a summary of your results.

4: Making a regional association plot

In this part of the practical, we will hone our R plotting skills to make beautiful regional association plots ('hitplots') of the most associated regions in our scan. This section of the practical is **optional** – you should only carry it out if you've successfully completed the other parts and have time to go on. A good hitplot shows several things:

- The evidence for association across the region
- Information about the recombination landscape and LD
- Regional genes and other potentially interesting annotations.
- Information on the variants being tested – such as whether they are directly typed or imputed (in this practical, we assume all variants are directly typed) or whether they have predicted function.

To make a hitplot we need to load data on all of these things. To begin, make sure you have loaded the association scan from the last part of the practical:

```
In RStudio:

data<-read.table("pccorrected-test.assoc.logistic",h=T)

data <- data[data$TEST == "ADD",]
```

4.1 Loading the data

We'll start by getting plink to compute LD between the most-associated SNP in each of the strongest three association peaks and the other SNPs in the dataset. First let's identify what the top SNPs in each region are:

```
In RStudio:

data <- data[order( data$P ), ]

data[ which( data$BP > 2.5E7 & data$BP < 3.5E7 )[1],]

data[ which( data$BP < 1.5E7 )[1],]
```

```
data[ which( data$BP > 4.5E7 )[1],]
```

Note down the rsids (in the ‘SNP’ column) of the three SNPs. As a sanity check, all three should have a P-value less than 10^{-6}. In this practical we’ll concentrate on the third SNP, which has the lowest P-value, but a question will ask you to repeat this for the other two SNPs. Now in the terminal let’s use plink to compute LD:

```
plink --bfile chr19-clean --keep resources/controls.txt --r2 dprime --
ld-window 10000 --ld-window-kb 1000000 --ld-window-r2 0.05 --ld-snps
rs57558361 rs376047 rs112820994
```

This command says to compute pairwise r^2 and D’ statistics between the three lead SNPs and all other SNPs within a window around them. Only SNPs with $r^2>0.05$ will appear in the output file. (We’ve restricted the computation to control samples. If these are population controls then we are estimating LD in the population). Let’s load the result into our R session:

```
In RStudio:
ld <- read.table( "plink.ld", hea=T, as.is=T )
View(ld)
```

Have a look at the results. The last two columns contain estimates of r^2 and D' between the SNPs listed in SNP_A and SNP_B columns. For plotting, it's simplest to bin those values:

```
In RStudio:
ld$R2_bin <- cut( ld$R2, breaks = seq( 0, 1, by = 0.1 ))
View(ld)
```

Now let's load the other data we need. First, we'll load information on genes from the file `resources/refGene_chr19.txt` (which comes from the UCSC Genome browser):

```
In RStudio:
genes <- read.table("resources/refGene_chr19.txt", hea=T, as.is=T)
```

```
View(genes)
```

This file is a bit complex. Among other things it contains gene names (in the `name2` column), transcription start and end points (`txStart`, `txEnd`), and the positions of exons (`exonStarts`, `exonEnds`). One interesting thing you might notice is that some genes (e.g. FUT2) have several transcripts. That's useful information but is a bit too much detail for our plot, so let's just keep the longest transcript of each gene:

In RStudio:

```
genes = genes[ order( genes$txEnd - genes$txStart, genes$exonCount,
decreasing = TRUE ), ]
genes = genes[ which( !duplicated( genes$name2 )), ]
```

We'll also load some helper code that will help us plot the genes:

In RStudio:

```
source( 'scripts/plot.genes.R' )
```

Next we'll load recombination rate estimates from the HapMap project:

In RStudio:

```
genetic_map <- read.table(
"resources/genetic_map_chr19_combined_b37.txt", hea=T, as.is=T )
View( genetic_map )
```

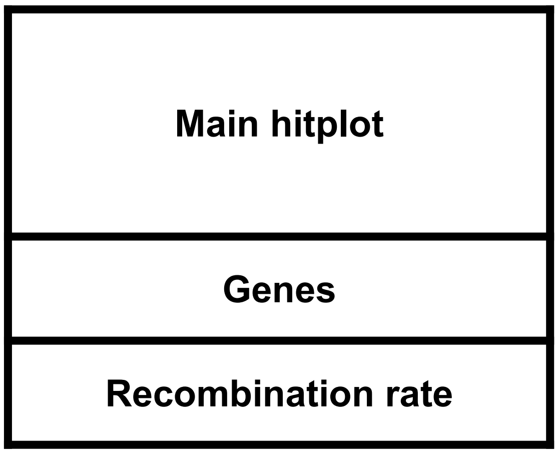
We're nearly ready to plot! We'll make a hit plot near the most-associated SNP (rs112820994). Let's find a region around it:

In RStudio:

```
focus.snp = 'rs112820994'
focus.pos = data$BP[ data$SNP == focus.snp ]
region = c( focus.pos - 250000, focus.pos + 250000 )
```

4.2 Making an initial hitplot

We'll make a plot divided into sections, something like this:



|

To do this, we'll use the `layout()` command in R to split the plotting area into sections with different heights.

```
In RStudio:

layout.matrix = matrix(c(1,2,3), ncol=1 )

layout( layout.matrix, heights = c(1,0.5,0.5) )
```

Now let's plot the association statistics. Since we're plotting other stuff underneath, we'll suppress the X axis label. We also use `par(mar = …)` to reduce the space around the plot.

```
In RStudio:

par( mar = c(1,4,2,2))

plot( data$BP, -log10( data$P ), xlim = region, xlab = '', ylab = "-
log10( P-value )" )
```

Now let's use the `plot.genes` function loaded above to plot genes in the second panel, and plot the recombination rate

```
In RStudio:

plot.genes( genes, region )

plot( genetic_map$position, genetic_map$ COMBINED_rate.cM.Mb., type =
'l', xlim = region, ylab = "cM/Mb" )
```

Beautiful! But not beautiful enough. Let's refine it.

4.2 Refining the hitplot

Now we'll make several modifications to make the plot really publication-ready. First we'll colour points using the binned r^2 values we loaded earlier. (We use the `heat.colors()` function to generate a nice palette for this).

```
In RStudio:
region.ld = ld[ ld$SNP_A == focus.snp & ld$BP_B >= region[1] &
ld$BP_B <= region[2] & !is.na( ld$R2_bin ), ]
data$colour = "black"
data$colour[ match( region.ld$SNP_B, data$SNP ) ] = heat.colors(10)
[ as.integer( region.ld$R2_bin ) ]
```

We can plot using these colours like this:

```
In RStudio:
par( mar = c( 1, 4, 2, 2 ))
plot( data$BP, -log10( data$P ), col = data$colour, pch = 20, xlim =
region, xlab = '', ylab = "-log10( P-value )" )
```

(In the above we used `pch=20` to get filled circles instead of open ones.)

It's also helpful to provide a legend. The legend() command can be used to add one to the plot. A legend with helpful labels is:

```
In RStudio:
legend( "topright", col=c("black",heat.colors(10)), pch = 20, legend
= c( "r²<0.1", sprintf( "r²≥%.1f",seq(0.1,0.9,by=0.1 ))))
```

We can also add a grid to let viewers locate points better:

```
In RStudio:

grid()
```

We have wrapped up the above code - with a few more cosmetic tweaks - into a function `hitplot()`, stored in the file `scripts/hitplot.R`. Load this into Rstudio and run it. You can use it to produce the finished plot like this:

```
In RStudio:

source( 'scripts/hitplot.R' )

hitplot('rs112820994', data, genes, ld, margin = 200000 )
```

Q. What gene is `rs112820994` in? Is it in an exon? Are other top SNPs in the region also in genes? (You may need to play around with the `margin` argument to zoom in or out of the plot. You can cross-check using the genome browsers).

Q. Are there any obvious features near the boundaries of the region of associated SNPs?

Q. Make a similar plot for each of the other regions we computed LD for above. Do they look similar? Is there anything odd about them?

Q. Having looked at the variants in the genome browser – what other genomic features would you add to the hitplot if you had time? How would you add these?

Q. (Challenge question – only try this if you've got time). In the sample folder there's a file called "`resources/snp138_chr19.txt.gz`" containing information from dbSNP on variants in each region. Load this in using `read.table()`. (You'll need to add the `sep='\t'` option to make it look for tab separators - type `?read.table` for help.) The column called `'func'` lists putative functional consequences of each variant. **The challenge is to**

create a version of the hit plot that overplots different shapes to show the functional annotation.

Hints:

1. The 'func' column in snp138 contains many categories and you'll first need to simplify it for plotting. One way is to take the most 'severe' category for each SNP. A rough list of useful categories might be: missense, nonsense, stop-gain, stop-loss, splice-3, splice-5, untranslated-3, untranslated-5, intron. E.g. the following code shows how to make a simpler variable func2:

```
snp138$func2 = NA
snp138$func2[grep('intron',snp138$func)] = 'intron'
snp138$func2[grep('splice-5',snp138$func)] = 'splice-5'
# …etc.
```

   To cut down copy-pasting, you may want to turn the above into a loop.

2. Use this to add a `'func2'` column to the association test data frame, `data`. You should be able to match snp138 to `data` using the `name` column. (This depends on the SNPs being named correctly though! Are there any SNPs that match by position but not name?)

3. The `points()` command can be used to overplot points on an existing plot. Use the `pch` argument to specify shape. (Only plot points that have `func != 'none')`.

4. `pch` takes argument that is a positive integer in the range 1-20, say (google for 'R shapes' to see a table of shapes). One way to turn categories into integers for `pch` is to first convert them to a factor, e.g.

```
data$func2 = factor(data$func2, levels = c( 'missense', 'nonsense',
'stop-gain', 'stop-loss', 'splice-3', 'splice-5', 'untranslated-3',
'untranslated-5', 'intron' )
```

```
table( as.integer(data$func2) )
```

5. You also need a legend!

Reference Information

You can download PLINK, and find much more information at the website:

https://www.cog-genomics.org/plink2

There is detailed documentation about all the options available, file formats and examples of commands.

A detailed tutorial (similar to work we have done here) is available at:

http://pngu.mgh.harvard.edu/~purcell/plink/tutorial.shtml