Quality Control for Genome Wide Association Studies

Cedric Gondro, Seung Hwan Lee, Hak Kyo Lee and Laercio R Porto-Neto

Summary

This chapter overviews the quality control (QC) issues for SNP-based genotyping methods used in genome wide association studies. The main metrics for evaluating the quality of the genotypes are discussed followed by a worked out example of QC pipeline starting with raw data and finishing with a fully filtered dataset ready for downstream analysis. The emphasis is on automation of data storage, filtering and manipulation to ensure data integrity throughput the process and on how to extract a global summary from these high dimensional datasets to allow better-informed downstream analytical decisions. All examples will be run using the R statistical programming language followed by a practical example using a fully automated QC pipeline for the Illumina platform.

Keywords: Genome wide association studies, quality control, Illumina, R statistics

1. Introduction

Data for genome wide association studies (GWAS) demand a fair amount of pre-processing and quality control (QC), especially SNP genotypes. A couple of good review articles on quality control issues in GWAS are given by Ziegler ***(1)*** and Teo ***(2).*** These are high dimensional data, which preclude any meaningful form of manual data evaluation. This means that any QC step will have to be automated with limited opportunity for the researcher to intervene directly in the process. The need for this high level of automation can lead to a suboptimal understanding of the dataset at hand and, while the objective of QC is to remove *bad* data points, there is a risk of adding additional bias through the process. In this chapter we will discuss the most commonly used QC metrics and show some simple code to run these analyses using R. Two underlying themes will run throughout the chapter; the first revolves around the importance of setting up a backbone infrastructure to ensure data integrity/consistency, and the second theme is on automating the QC steps, and also summarizing these results into a *human digestible* format which will allow the data to reveal itself and help guide decisions on the best way forward for downstream analysis. A fully automated pipeline for analysis and reporting of QC results for Illumina SNP data is available at http://www-personal.une.edu.au/~cgondro2/CGhomepage. This pipeline is briefly discussed at the end of the chapter.

2. Platform

In recent years R ***(3)*** has become *de facto* statistical programming language of choice for statisticians and it is also arguably the most widely used generic environment for analysis of high throughput genomic data. We will use R to illustrate the concepts and show how to implement the QC metrics in practice, but it is straightforward to port them to other platforms. Herein we assume the reader is

reasonably familiar with R and its syntax. For those who are unfamiliar with it, two excellent texts more focused on the programming aspects of the language are Chambers ***(4)*** and Jones *et al.* ***(5).***

3. Storing and handling data

Data management is an important step in any project, but it is particularly critical with large datasets. Unfortunately it is often relegated to second plan. A common workflow in genome wide association studies starts with a phenotypes and samples collection stage, followed by SNP genotyping, data QC, genotypes phasing, imputation and actual association testing. At each point data gets changed (e.g. SNP and samples are discarded in the QC stage); it is paramount to be able to track these changes and revert back to the original raw data at any stage, if things go wrong – and they rather frequently tend to .

Databases are by far the best approach to manage large datasets. While for large collaborative projects it is essentially mandatory to have a dedicated database manager to design the database and manage data storage/access, it is still easy to implement robust solutions for smaller projects. To illustrate we will work with a small simulated dataset that can be downloaded from the book's website. This is a small dataset with a limited number of samples and only 50,000 SNP but still convenient to illustrate the process. This data could quite easily be handled directly in R, but for larger datasets dimensionality can become a problem - the whole dataset would not fit into the memory of common desktop machines. The simplest work around is to store the data in a database and retrieve only the parts of the data that are needed at any given time. R can interface quite easily with databases – SQL queries can be sent straight from R to the database and retrieved records stored as a *data frame*.

SNP array data will usually come in two formats: either in a proprietary database structure developed by the chip manufacturer (e.g. the Genome Studio *bim* files) or as a flat file. We will work only with the flat files as it is the most commonly used form. The first step is to build a database from the flat files for further downstream analysis. This can be built straight from R, but of course it does not have to be. There are many options for working with databases. We will use *SQLite*. The key advantages are that it connects well to R – e.g. the annotation packages from Bioconductor (www.bioconductor.org) are all built with it; there's no installation involved – a single 500kb executable is all that's needed; databases can simply be copied across machines without any further installation. To access an SQLite database from R the *RSQLite* package is needed.

For our example we will use two files: a *genotypes* file and a *map* file. The first contains all genotype calls for all SNP and all samples (Figure 1); the second holds mapping information of the SNP, e.g. chromosome, physical location, etc.

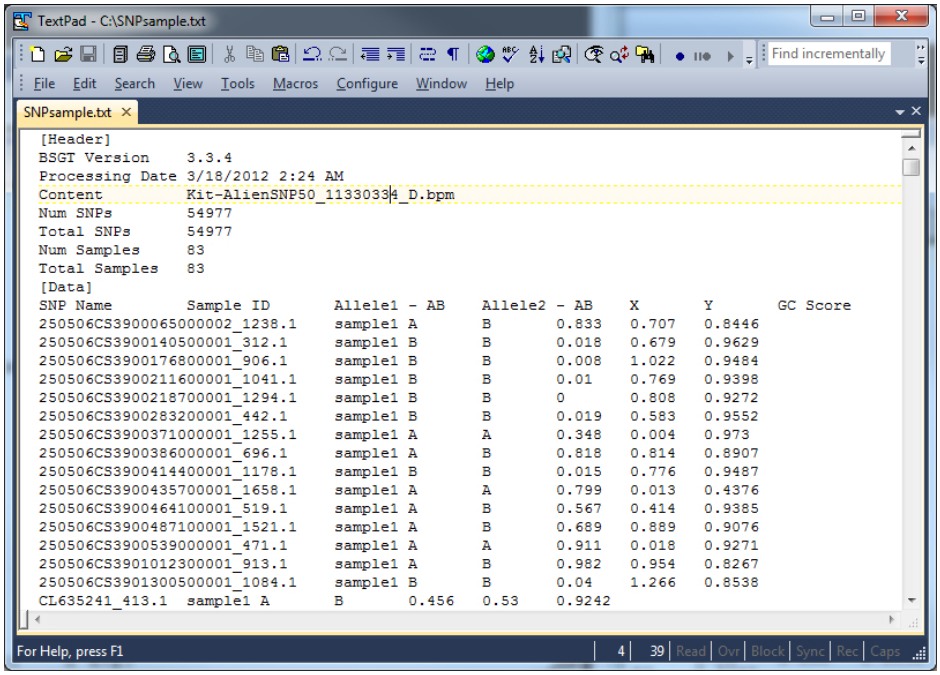

Figure 1. Example Illumina SNP genotype file exported from GenomeStudio.

To build a database directly in SQLite we first need to create a schema (a schema is simply a text file that describes the database structure and is used to create the tables and fields in an empty DB) with the tables and columns we want in it. A schema can be written in any plain text editor. Here the *map* file has only 3 columns: *SNP name, chromosome and position* but could have additional columns, e.g. actual SNP nucleotides. The *genotypes* file (Figure 1) has 7 columns: *SNP name*, *Sample ID*, *Allele1*, *Allele2*, *X*, *Y* and *GC Score* (the first 4 columns are intuitive and the last three we will discuss later on). Notice that we have usual nuisance information lines (header text), 9 lines in this example. A simple schema for these data will consist of two tables, one for each of the files and one column for each source of information. A *snpmap* table with SNP name, chromosome and position, and a *snps* table with the 7 columns from our dataset. The schema could look like

```
CREATE TABLE snpmap(
  name,
  chromosome,
  position
);

CREATE TABLE snps(
  snp,
  animal,
  allele1,
  allele2,
  x,
  y,
  gcscore
);

CREATE INDEX snp_idx ON snps(snp);
```

```
CREATE INDEX animal_idx ON snps(animal);
CREATE INDEX chromosome_idx ON snpmap(chromosome);
```

This is simply a plain text file with the structure (description) of the DB that we want to create. The table and columns names are discretionary; here we decided to hold mapping information (3 columns) in *snpmap* and the genotypes in *snps* (7 columns). Note that indices are created at the end to make sure that searches by SNP ID, sample ID or chromosome are fast to execute (indices make the database slow to build but speed up the queries dramatically). A good source of information for SQLite syntax can be found at http://www.sqlite.org/. Once we've saved our schema in a text file (e.g. *snpDB.sql* or *schema.txt*) we are ready to create the database and populate it with the data. This can be done on the command line (provided of course *sqlite* is installed and in the OS path) with:

```
sqlite SNPsmall < snpDB.sql
```

and this creates a new database "`SNPsmall`" with the previous schema ready to be populated with the genotype data. It is also simple to create a new database directly in R, and for this case not even the schema is needed. First open R and load the *RSQLite* package

```
library(RSQLite)
```

Then run the following code:

```
con=dbConnect(dbDriver("SQLite"), dbname = "SNPsmall")

dbWriteTable(con,"snpmap","SNPmap.txt", header= TRUE, append=T,
sep="\t")

dbWriteTable(con,"snps","SNPsample.txt", append=T, header=TRUE,
skip=9,sep="\t")
```

Let's go over the code. The first line simply creates a new blank database called *SNPsmall* using the *SQLite* database driver (R has DBMS's for most common database engines). In the same line a connection (function *dbConnect*) to the new database is created (if the database already existed, R would simply open a connection to it).

Now we have an empty DB similar to what we did using straight *sqlite* but without any tables/fields information. R can simply create tables and fields directly from the flat files themselves. To populate the DB (once connected to it using *dbConnect*) we can upload our flat files of genomic data *SNPmap.txt* and *SNPsample.txt* using the function *dbWriteTable* (once for each file). The function takes quite a few arguments: the database connection *con* that was created in the previous line, the name of the a new table to be created in the DB, the name of the flat file to import, if there is a header in the file it can be used to create the field/column names, if appending or not to the database, the separator used between columns in the data and how many lines to skip if there are extraneous header files. Note that in this example the top lines from the genotypes file have to be removed (Figure 1).

We can have a look at the tables and fields in the DB using *dbListTables* and *dbListFields*

```
dbListTables(con)
> "snpmap" "snps"

dbListFields(con,"snpmap")
> "name" "chromosome" "position"

dbListFields(con,"snps")
> "snp" "animal" "allele1" "allele2" "x" "y" "gcscore"
```

The function *dbGetQuery* is used to send an SQL query to the DB and return the data in a single step. A two-step approach is using *dbSendQuery* and fetch, but we will not discuss these here. The syntax for *dbGetQuery* is *dbGetQuery (connection name, "SQLquery").* For example, to retrieve the number of records in a table:

```
dbGetQuery(con,"select count (*) from snpmap")
> 1     54977

dbGetQuery(con,"select count (*) from snps")
> 1   4563091
```

That looks about right. There are 54,977 records in *snpmap* and we know the chip has 54,977 SNP so that matched up well. The number of records in *snps* is also fine - it should be the number of samples times the number of SNP. To retrieve sample ids we would do something along the lines of

```
animids=dbGetQuery(con, "select distinct animal from snps")

dim(animids)
> 83  1

head(animids)

  animal
1 sample1
2 sample10
3 sample11
4 sample12
5 sample13
6 sample14
```

Herein we will not discuss details of SQL queries or syntax. Any general SQL book will cover most of the common needs (see for example ***(6)***). All we really need to know is how to use *select* * from *tableName* where *columnName="mysearch"*. For example to retrieve all data associated to the first sample.

```
animids=as.vector(animids$animal)
hold=dbGetQuery(con,paste("select * from snps where
animal='",animids[1],"'", sep=""))

dim(hold)
```

```
> 54977     7

head(hold)

                           snp  animal allele1 allele2
1 250506CS3900065000002_1238.1 sample1       A       B
2  250506CS3900140500001_312.1 sample1       B       B
3  250506CS3900176800001_906.1 sample1       B       B
4 250506CS3900211600001_1041.1 sample1       B       B
5 250506CS3900218700001_1294.1 sample1       B       B
6  250506CS3900283200001_442.1 sample1       B       B
      x     y gcscore
1 0.833 0.707  0.8446
2 0.018 0.679  0.9629
3 0.008 1.022  0.9484
4 0.010 0.769  0.9398
5 0.000 0.808  0.9272
6 0.019 0.583  0.9552
```

In the first line we just changed the *data.frame* with animal ids to a vector - saves some indexing work. Then we retrieved the data for the first sample from our vector of sample ids. It's quite easy to picture a loop for each sample - read in all genotypes for the sample, run some QC metrics, read in the next sample... Notice the use of *paste* to create a query string and also the rather awkward use of single and double quotes - we need quotes for the R string and we also need to include a single quote for the SQL query in the DB. Just the query string looks like

```
paste("select * from snps where animal='",animids[1],"'",sep="")

> "select * from snps where animal='sample1'"
```

We already have a vector for the samples. Let's also get a vector of SNP.

```
snpids=as.vector(dbGetQuery(con, "select distinct name from
snpmap")[,1])

length(snpids)
> 54977

head(snpids)
[1] "250506CS3900065000002_1238.1"
[2] "250506CS3900140500001_312.1"
[3] "250506CS3900176800001_906.1"
[4] "250506CS3900211600001_1041.1"
[5] "250506CS3900218700001_1294.1"
[6] "250506CS3900283200001_442.1"
```

And one last thing. When finished with the DB we should close the connection.

```
dbDisconnect(con)
```

4. Quality control

Now that the data is safely tucked away in a database, we are ready to do some quality control on the genotypes. Various metrics are commonly used and there is still some level of subjectivity in these, particularly when setting thresholds. The statistics are performed either across SNP or across samples. The key concept is to detect SNP and/or samples that should be removed prior to the association analyses. Let's start with across SNP analyses.

4.1 Genotype calling and signal intensities

SNP alleles are usually coded as A/B in Illumina or the actual nucleotides are used. This changes depending on the platform or laboratory, but preference should be to use a simple reference for alleles and an additional DB table with all pertinent information for the SNP. Let's have a look at the first SNP (*snpids[1]*) in the dataset.

```
con=dbConnect(dbDriver("SQLite"),dbname = "SNPsmall")

snp=dbGetQuery(con,
     paste("select * from snps where snp='", snpids[1],"'",sep=""))

dim(snp)
 > 83  7

head(snp)
                            snp  animal allele1 allele2
1 250506CS3900065000002_1238.1 sample1       A       B
2 250506CS3900065000002_1238.1 sample5       A       B
3 250506CS3900065000002_1238.1 sample6       A       B
4 250506CS3900065000002_1238.1 sample7       B       B
5 250506CS3900065000002_1238.1 sample8       B       B
6 250506CS3900065000002_1238.1 sample9       B       B
      x     y gcscore
1 0.833 0.707  0.8446
2 0.829 0.714  0.8446
3 0.816 0.730  0.8446
4 0.031 1.132  0.8446
5 0.036 1.146  0.8446
6 0.037 1.150  0.8446

snp$allele1=factor(snp$allele1)
snp$allele2=factor(snp$allele2)

summary(snp$allele1)
>  A  B
> 36 47

summary(snp$allele2)
> A  B
> 6 77
```

First reopen the DB connection and then send an SQL query to retrieve data for the first SNP. Convert alleles into factors (usually data is returned as *character*) and then summarize the allele information. There are no missing values in the data and as expected, there are only two alleles - A

and B. If there were missing values (missing calls, to use the terminology) we would see a third factor (e.g. NA or commonly "-"). Our data also has an *X* and a *Y* column (this is for the specific case of Illumina data). These are the normalized intensities of the reads for each of the two alleles. Allele calls are assigned based on the signal intensity of the fluorescence read by the scanner. These intensities can be plotted as an *XY* plot. We would expect that one of the homozygous genotypes would show high X values and low Y values while the other homozygote would be the opposite. Heterozygotes would be somewhere between the two. If the technology was 100% accurate there would be only 3 perfect points on the plot and all samples would have the same intensity measures; but since reality gets in the way, what we do observe are 3 clouds (clusters) of data which will hopefully separate well between each other (Figure 2). To plot the data:

```
snp=data.frame(snp, genotype=factor(paste(snp$allele1,
snp$allele2,sep=""), levels=c("AA","AB","BB")))

plot(snp$x,snp$y,col=snp$genotype, pch=as.numeric(snp$genotype),
 xlab="x",ylab="y", main=snp$snp[1],cex.main=0.9)

legend("bottomleft",paste(levels(snp$genotype),"(",summary(snp$genot
ype),")",sep=""), col= 1:length(levels(snp$genotype)), pch=
1:length(levels(snp$genotype)), cex=0.7)
```

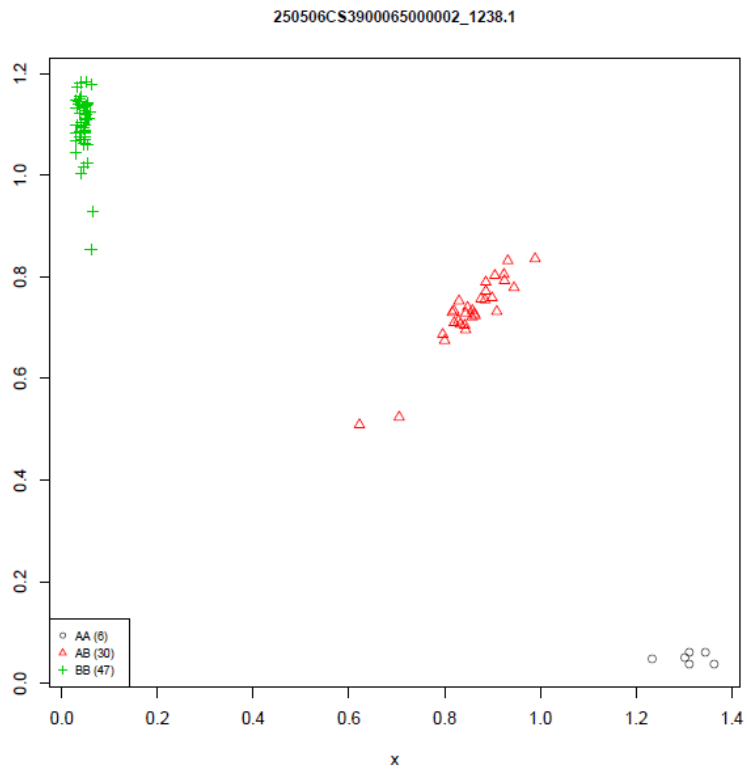

Figure 2. Clustering of genotype calls based on X/Y coordinates.

The genotypes were colour coded and symbols were used to make it easier to distinguish between them. Notice how the genotypes clearly cluster into three discrete groups - an indication of good data. Of course it is not possible to look at each of these plots for every SNP. Common practice is to go back to these plots after the association test and make sure that the significant SNP have clear

distinction between clusters. There are some methods to summarize the clusters into an objective measurement e.g. sums of the distances to the nearest centroid of each cluster and the individual calls themselves.

Another metric included in this dataset is the GC score, without any in-depth details, it is a measure of how reliable the call is (essentially, distance of the call to the centroid as we mentioned above) on a scale from 0 to 1. Some labs will not assign a call (genotype) to GC scores under 0.25. Another common rule of thumb number is to cull reads under 0.6 (and projects working with human data may use even higher thresholds of 0.7-0.8).

```
length(which(snp$gcscore<0.6))
> 0
```

For this particular SNP all GC scores are above 0.6. Figures 3 and 4 exemplify what a good and a bad SNP look like. We might want to cull individual reads based on a threshold GC score value, but we might also remove the whole SNP if, for example, more than 2% or 3% of its genotyping failed or if the median GC score is below a certain value (say 0.5 or 0.6). Again, the SNP we are analysing is fine.

```
median(snp$gcscore)
> 0.8446
```

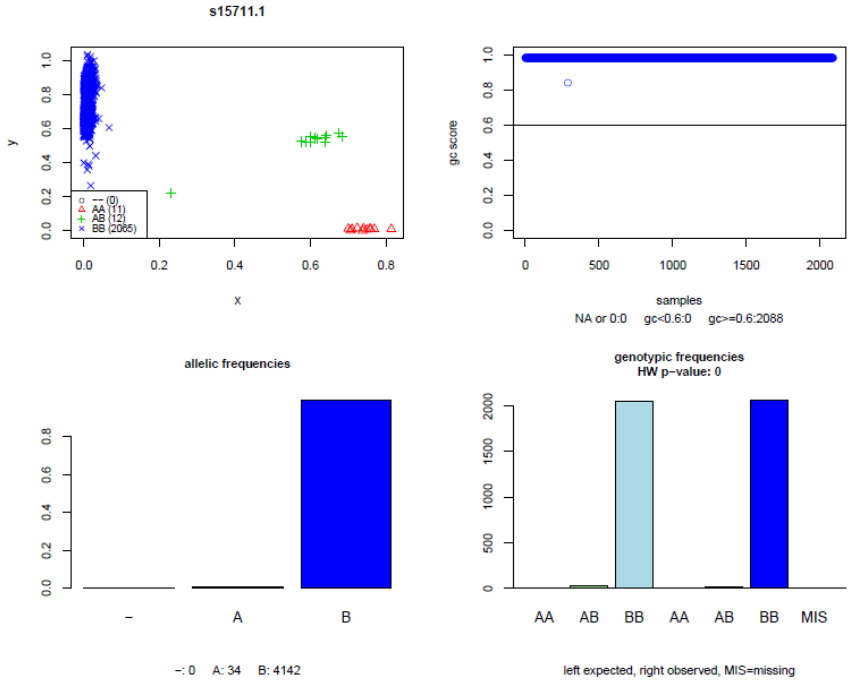

Figure 3. Example of a good quality SNP. Top left: clustering for each genotype (non-calls are shown as black circles). Top right: GC scores. Bottom left: non-calls and allelic frequencies (actual counts are shown under the histogram). Bottom right: genotypic counts, on the left hand side the expected counts and on the right the observed counts; the last block shows number of non-calls.

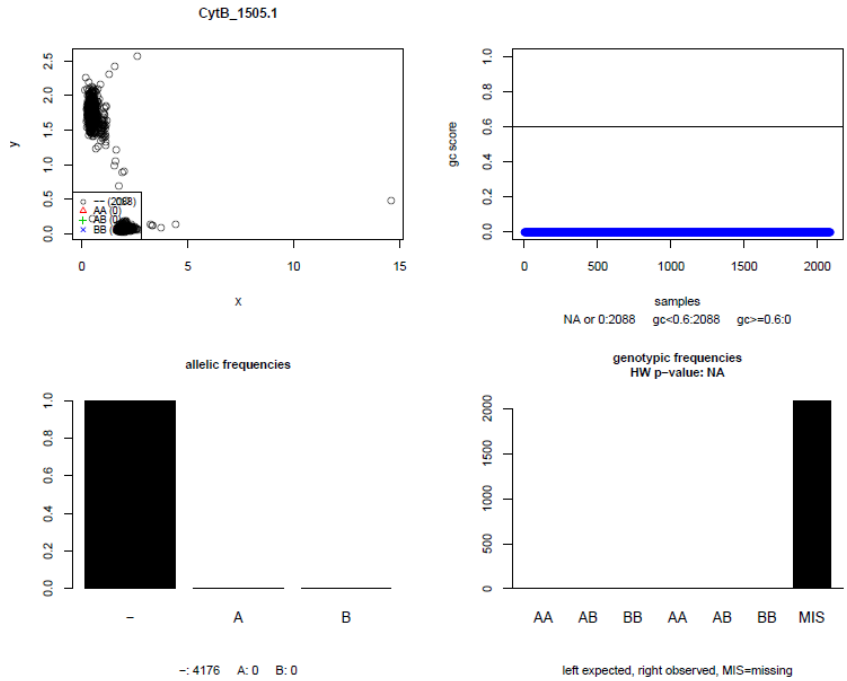

Figure 4. Example of a bad quality SNP. Top left: clustering for each genotype (non-calls are shown as black circles - here all samples). Top right: GC scores. Bottom left: non-calls and allelic frequencies (actual counts are shown under the histogram). Bottom right: genotypic counts, on the left hand side the expected counts and on the right the observed counts; the last block shows number of non-calls.

4.2 Minor allele frequency and Hardy-Weinberg equilibrium

Population based metrics are also employed. A simple one is the minor allele frequency (MAF). Not all SNP will be polymorphic, some will show only one allele across all samples (monomorphic) or one of the alleles will be at a very low frequency. The association between a phenotype and a rare allele might be supported by only very few individuals (no power to detect the association), in this case the results should be interpreted with caution. To avoid this potential problem, SNP filtering based on MAF is often used to exclude low MAF SNP (usual thresholds are between 1% and 5%), but it is worthwhile to check the sample sizes and estimate an adequate value for your dataset. Back to the example SNP the allelic frequencies are

```
alleles=factor(c(as.character(snp$allele1),
as.character(snp$allele2)),levels=c("A","B"))

summary(alleles)/sum(summary(alleles))*100

> A        B
25.3012 74.6988
```

The frequencies are reasonable, around one-quarter *A* allele and three-quarters *B* allele. But again the point to consider is the objective of the work and the structure of the actual data that was collected. For example, if QC is being performed on mixed samples with an overrepresentation of one group, it is quite easy to have SNP that are not segregating in the larger population but are segregating in the smaller one – the MAF frequency in this case will essentially be the proportion of the minor allele from the smaller population in the overall sample. And if the objective of the study

was to characterize genetic diversity between groups, the interesting SNP will have been excluded during the QC stage.

The next metric is Hardy-Weinberg equilibrium - HW. For a quick refresher, the Hardy-Weinberg principle, independently proposed by G. H. Hardy and W. Weinberg in 1908, describes the relationship between genotypic frequencies and allelic frequencies and how they remain constant across generations (hence also referred to as Hardy-Weinberg equilibrium) in a population of diploid sexually reproducing organisms under the assumptions of random mating, an infinitely large population and a few other assumptions.

Consider the bi-allelic SNP with variants A and B at any given locus, there are three possible genotypes: AA, AB and BB. Let's call the frequencies for each genotype D, H and R. Under random mating (assumption of independence between events) the probability of a cross AA x AA is D^2, the probability for AB x AB is 2DH and the probability for BB x BB is R^2. If p is the frequency of allele A (p=D+H/2) then the frequency of B will be q=1-p and consequently the genotypic frequencies D, H and R will respectively be p^2, 2pq and q^2. This relationship model in itself is simply a binomial expansion.

Hardy-Weinberg equilibrium can be seen as the null hypothesis of the distribution of genetic variation when no biologically significant event is occurring in the population. Naturally real populations will not strictly adhere to the assumptions for Hardy-Weinberg equilibrium, but the model is however quite robust to deviations. When empirical observations are in a statistical sense significantly different from the model's predictions, there is a strong indication that some biologically relevant factor is acting on this population or there are genotyping errors in the data. This is where HW becomes controversial - it can be hard to distinguish a genotyping error from a real population effect, particularly when dealing with populations from mixed genetic backgrounds. Common p-value thresholds for HW are e.g. 10^{-4} or less (in practice use multiple testing corrected p-values, so much lower cut offs). To calculate HW for a SNP in R:

```
obs=summary(factor(snp$genotype, levels=c("AA","AB","BB")))

> AA AB BB
   6 30 47

hwal=summary(factor(c(as.character(snp$allele1),
as.character(snp$allele2),levels=c("A","B"))))

hwal=hwal/sum(hwal)

     A         B
0.2559524 0.7440476

exp=c(hwal[1]^2,2*hwal[1]*hwal[2],hwal[2]^2)*sum(obs)
names(exp)=c("AA","AB","BB")
# chi-square test
# with yates correction

xtot=sum((abs(obs-exp)-c(0.5,1,0.5))^2/exp)

# get p-value
```

```
pval=1-pchisq(xtot,1)
> 0.8897645
```

It is a typical χ-square test. The only additional part is the Yates correction used when adding up the χ-square values. And, in this example, the SNP is in Hardy-Weinberg equilibrium.

4.3 Call rates and correlations in samples

Quality control across samples is similar to what was done with the SNP. If 2% or 3% of the genotypes are missing (call-rate <0.97) it is probably a good idea to exclude the sample, it is an indication of poor DNA quality. Another criterion that has not been discussed so far is the correlation between samples. If samples show very high correlations they might have to be excluded. Non related samples would on average show a correlation of 0.5. Of course, again, the structure of the data has to be taken into account - a case-control study with random samples is very different from a half-sib project in livestock. In R correlations are computationally intensive since all data needs to be stored in memory as a matrix to calculate the pairwise correlations (this can be split into pairs but will be painstakingly slow for larger datasets). The R function is simply *cor(NameOfGenotypesMatrix)*. If you cannot fit the data matrix into memory consider some workarounds such as estimating correlations from a manageable subset of the SNP at a time.

4.4 Heterozygosity

Another useful metric is heterozygosity, which is simply the proportion of heterozygotes in relation to all genotypes. It is also worthwhile to check heterozygosity on SNP and compare to the expected heterozygosity (or gene diversity), it's just more common to evaluate heterozygosity on the samples. Essentially if a sample's heterozygosity is too high it can be an indication of DNA contamination (and once again: it could also be simply that a small proportion of samples are *truly* very different from the bulk of the data). Removal of samples that depart plus or minus 3 standard deviations from the mean is a reasonable approach.

We will need the heterozygosity for all samples before we can look for outliers. We could just go over all samples in the DB, calculate (and store) the heterozygosity for each subject and discard the data from memory. You might have to do that if the dataset is too large, but since the example set is quite small let's build a matrix with the genotype counts (*sumslides*) for all samples and a matrix of SNP x sample (*numgeno*) – this is the entire dataset.

```
sumslides=matrix(NA,83,4)
rownames(sumslides)=animids
colnames(sumslides)= c("-/-","A/A","A/B","B/B")

# hold reshaped (numeric data)
numgeno=matrix(9,54977,83)

for (i in 1:83)
{
  hold=dbGetQuery(con,
  paste("select * from snps where animal='",animids[i],"'",sep=""))

  hold=data.frame(hold,
       genotype=factor(paste(hold$allele1,hold$allele2,
       sep=""),levels=c("--","AA","AB","BB")))
```

```
  hold=hold[order(hold$snp),]

  sumslides[i,]=summary(hold$genotype)

  temp=hold$genotype
  levels(temp)=c(9,0,1,2)

  numgeno[,i]=as.numeric(as.character(temp))

  # change to 9 genotypes under GC score cutoff
   numgeno[which(hold$gcscore<0.6),i]=9
 }

rownames(numgeno)=hold$snp
colnames(numgeno)=animids

head(sumslides)

         -/-   A/A   A/B   B/B
sample1  838 15818 20100 18221
sample10 777 15397 21367 17436
sample11 803 15381 21564 17229
sample12 763 15440 21145 17629
sample13 822 16257 19524 18374
sample14 750 15637 21014 17576


numgeno[1:10,1:3]
                              sample1 sample10 sample11
250506CS3900065000002_1238.1       1        2        1
250506CS3900140500001_312.1        2        1        2
250506CS3900176800001_906.1        2        1        2
250506CS3900211600001_1041.1       2        2        2
250506CS3900218700001_1294.1       2        2        1
250506CS3900283200001_442.1        2        0        1
250506CS3900371000001_1255.1       0        2        2
250506CS3900386000001_696.1        1        1        0
250506CS3900414400001_1178.1       2        2        2
250506CS3900435700001_1658.1       9        9        9
```

We defined a matrix to store genotype counts for each sample (*sumslides*) and gave names to the rows and columns just to make it easier to identify in the output (see above). Notice that we used three genotypes plus -/- for missing genotypes. Another matrix *numgeno* was created to store all genotypic data. Then we made a loop to query the DB and extract data for each animal, sorted the data by SNP (using order) to make sure that the data returned by the DB is always in the same order; summarized the genotypic data in their classes and added the results to *sumslides*. Then we re-levelled the genotypes into numeric format (9 - missing, 0 - AA, 1 - AB, 2 - BB), this simply because it is a common format for downstream analysis. And in the last line of the loop we set all genotypes with GC scores under 0.6 as missing. Finally some housekeeping, assign names to the rows and columns so we can identify SNP and samples and check if everything looks alright.

To calculate and plot heterozygosities is quite simple now that we have the data. All that's needed is to divide the number of heterozygotes by the total genotypes; calculate the mean and standard deviation and then calculate the values for 3 standard deviations (3SD) to each side of the mean. Finally plot (Figure 5) the data and add lines for the mean and 3SD (the SD lines do not show up in the figure since there are no outliers in our data).

```
samplehetero=sumslides[,3]/(sumslides[,2]+
sumslides[,3]+sumslides[,4])

# outliers 3 SD
up=mean(samplehetero)+3*sd(samplehetero)
down=mean(samplehetero)-3*sd(samplehetero)
hsout=length(which(samplehetero>up))

# number of outliers
hsout=hsout+length(which(samplehetero<down))

plot(sort(samplehetero),1:83,col="blue",cex.main=0.9,
     cex.axis=0.8,cex.lab=0.8,
     ylab="sample",xlab="heterozygosity",
     main=paste("Sample heterozygosity\nmean:",
     round(mean(samplehetero),3),"    sd:",
     round(sd(samplehetero),3)),
     sub=paste("mean: black line     ",3,
     "SD: red line     number of outliers:",hsout),
     cex.sub=0.8)

abline(v=mean(samplehetero))
abline(v=mean(samplehetero)-3*sd(samplehetero),col="red")
abline(v=mean(samplehetero)+3*sd(samplehetero),col="red")
```

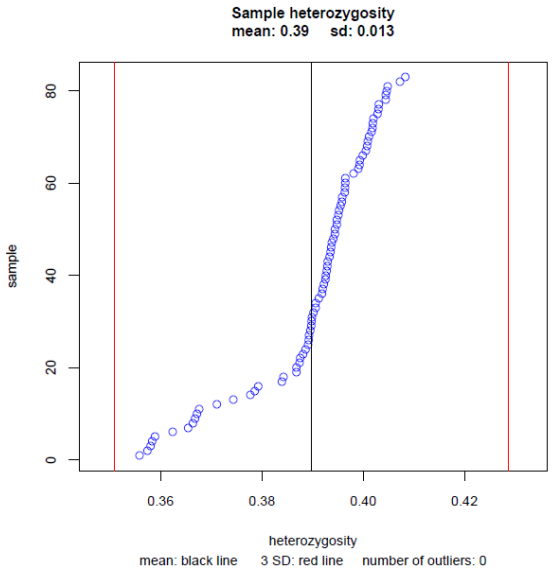

Figure 5. Distribution plot of samples heterozygosity. All samples are within 3 standard deviations from the mean.

We still have not calculated the correlation matrix. The function *cor* will only work with numeric data, hence us changing the genotypes to numeric format (it's also much smaller to store - 10M x 208M in the original file).

```
animcor=cor(numgeno)

library("gplots")
hmcol=greenred(256)
heatmap(animcor,col=hmcol,symm=T,labRow=" ",labCol=" ")
```

The first line calculates the correlation matrix, a simple Pearson correlation, and the remaining lines of code are used to plot the results as a heatmap (Figure 6). Heatmaps are excellent to visualize relationships between data. The library *gplots* also has some nice graphing capabilities. Note that missing data was replaced by 9 – this greatly inflates differences between samples and, on the other hand, strongly pulls together samples with a lot of missing data. Keep in mind that this is for QC purposes only; such an approach should not be used to estimate genomic relationships from the data.

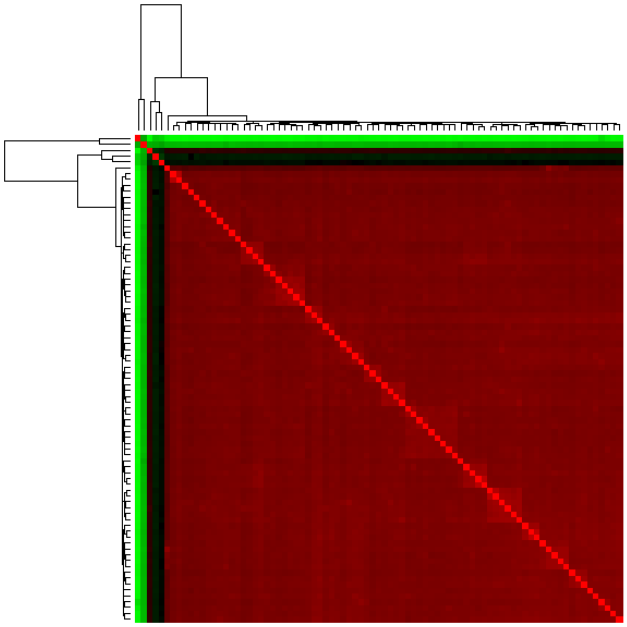

Figure 6. Heatmap of correlations between samples. Samples on the outer edges are very different from the bulk of the data. A strong indication of bad quality samples.

A couple of last comments: 1) what was discussed here was across all SNP and/or samples. With case-control studies it is worth considering running these metrics independently on cases and controls and then checking the results for consistency. 2) We have not plotted any results based on mapping information; it is a good idea to plot e.g. HW statistics per chromosome to see if there are any evident patterns such as a block on the chromosome that is consistently out of HW.

5. Fully automated QC for Illumina SNP data

In the book's website there is a full example of QC report for the dataset used in this chapter. The entire report is automatically generated using an R program and a full dataset of data filtered

applying the QC metrics is also output for further analyses and summarized. This way of viewing the data is preferable to simply applying filtering without investigating the actual data structure. Once all metrics are summarized and pulled together into a report it becomes much easier to understand what each of these metrics are doing to the data and it also provides a chance to QC the QC itself.

The program also builds a database with the genotypic data and at the end of the run adds the QC results as additional tables to the database for future reference or fine tuning of filtering parameters based on an evaluation of the QC report. The program and documentation and an example dataset is freely available for download from http://www-personal.une.edu.au/~cgondro2/CGhomepage.

Acknowledgements

This work was supported by a grant from the Next-Generation BioGreen 21 Program (No. PJ008196), Rural Development Administration, Republic of Korea.

References

1. Ziegler, A., I. R. Konig, and J. R. Thompson (2008). *Biostatistical Aspects of Genome-Wide Association Studies.* Biometrical Journal of Statistical Software, 50, 8-28.

2 . Teo, Y. Y. (2008). *Common statistical issues in genome-wide association studies: a review on power, data quality control, genotype calling and population structure*. Current Opinion in Lipidology, 19, 133-143.

3. R Development Core Team (2012) *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria

4. Chambers, J. M. (2008). *Software for Data Analysis: Programming with R*. Springer.

5. Jones, M., R. Maillardet and A. Robinson (2009). *Scientific Programming and Simulation using R*. CRC Press.

6. Ramalho, J. A. (2000). *Learn SQL.* Wordware Publishing.