User's Manual for ASRgenomics v. 1.1.0

An R package with complementary genomic functions

VSNi

Prepared by

Salvador A. Gezan, Amanda A. de Oliveira
Giovanni Galli and Darren Murray

May 26, 2022

User's Manual for ASRgenomics v. 1.1.0

Bibliographical reference

Gezan, S.A., de Oliveira, A.A., Galli, G., and Murray, D. 2022. ASRgenomics: An R package with Complementary Genomic Functions. Version 1.1.0. VSN International, Hemel Hempstead, United Kingdom.

Authors' email addresses

Salvador A. Gezan salvador.gezan@vsni.co.uk
Amanda A. de Oliveira support@vsni.co.uk
Giovanni Galli giovanni.galli@vsni.co.uk
Darren Murray darren.murray@vsni.co.uk

Companion resources

This R library and associated documentation can be downloaded from:
https://vsni.co.uk/free-software/asrgenomics

Acknowledgements

The authors gratefully acknowledge the team of testers and evaluators from of our tool for helping make this package more useful and robust.

Preface

`ASRgenomics` is a package that presents a series of molecular and genetic routines in the `R` environment with the aim of assisting in analytical pipelines before and after the use of `ASReml-R` or another library to perform analyses such as Genomic Selection (GS) or Genome-Wide Association Studies (GWAS).

The main tasks considered are:

- Preparing and exploring pedigree, phenotypic and genomic data.
- Calculating and evaluating genomic matrices and their inverse.
- Complementing and expanding results from genomic analyses.

The functions implemented consider aspects such as: filtering SNP data for quality control; assessing a kinship matrix by reporting diagnostics (statistics and plots); performing Principal Component Analysis (PCA) based on kinship or SNP matrices for understanding population structure; calculating the genomic matrix **G** and its inverse and assessing their quality; matching pedigree- against genomic-based matrices; tuning up a genomic matrix (bend, blend or align); and obtaining the hybrid matrix **H** as required for single-step GBLUP (ssGBLUP).

The routines presented here are the product of years developing code and applying these tools to genomic data from animal and plant breeding programs. The initial concept originated from Nazarian and Gezan (2016) who considered some of these elements in their software `GenoMatrix`, but this has been extended further in the `R` environment (R Core Team 2020) by considering additional statistical routines and elements developed during the last few years.

The intent of this tool is to facilitate the performance of genomic analyses such as GS and GWAS, in a straightforward and efficient manner, along with providing full replicability to these analyses. Our aim is that these functions can be easily used with `ASReml-R` (Butler et al. 2009) to fit linear mixed models (LMMs). However, these LMMs can be fitted in any other software, library or routine of your choice.

We have attempted to consider some of the latest publications and developments, but this is in no way comprehensive, especially given the dynamics of this field with a constant flow of new analytical improvements and developments.

In this manual, we will present some of the structure of `ASRgenomics` illustrating its capabilities with examples using datasets provided in this package.

`ASRgenomics` is a tool that is provided "as is". We have made all efforts to check our routines carefully and we have used real, publicly available datasets as much as possible, and in cases with limited information, we have considered some simplifying assumptions.

Contents

1 Getting Started

ASRgenomics is an R package available for Linux, Windows and Mac OS that can be downloaded from https://vsni.co.uk/free-software/asrgenomics

Download the appropriate version of ASRgenomics for your operating system.

- For Windows, the download will be a .zip file.
- For Linux, the download will be a .tgz file.
- For Mac, the download will be a .tar.gz file.

Before installing ASRgenomics you will need to install the following packages:

- AGHmatrix
- cowplot
- crayon
- data.table
- ellipse
- factoextra
- ggplot2
- scattermore
- superheat

These are available for installation from the CRAN website https://cran.r-project.org/. Start an R session and install the packages above.

Install ASRgenomics using one of the following commands as appropriate for your operating system.

For Windows:

```r
install.packages(path, repos = NULL, type = "win.binary")
```

For Linux:

```r
install.packages(path, repos = NULL)
```

For Mac:

```r
install.packages(path, repos = NULL, type = "source")
```

where `path` is the location and name of the appropriate file for your operating system to install, for example: `"/home/<yourusername>/ASRgenomics1.1.0.zip"`.

Another option is to install `ASRgenomics` directly from `RStudio` by first going to the menu: `>Tools/InstallPackages...`, in the `Install from` select

`"Package Archive File (.zip; .tgz; tar.gz)"`

and then you will have to search for the location of the file on your computer and select it. Finally, you just need to click on `Install`.

Once installed, load the library using the command:

```
library(ASRgenomics)
```

Now you are ready to use it! A good way to get started is to request help directly from this library. For example, you can type:

```
help("G.inverse")
```

A complete description of the functions and the datasets used in this manual are given in the `ASRgenomics` *Package Reference* or directly from the help pages available in `R` by typing:

```
help(ASRgenomics)
```

Once `ASRgenomics` is loaded, you can also access the datasets by using the `data()` function, for example:

```
data(geno.apple)
```

In the next sections we will present how to prepare and run genomic analyses for an array of different cases and situations.

2 Analytical Genomics Flow with ASRgenomics

Performing statistical analyses with molecular data, in the context of GS or GWAS, using data from operational breeding programs can be a difficult task. Traditional genetic analyses using only pedigree data requires that the phenotypic data and pedigree communicate properly. This implies, for example, that all individuals present in the phenotypic data have been included in the kinship matrix. Once we add molecular information to our genetic analyses, then we have an additional layer of complexity, as now we need to manage the molecular data to communicate properly with our phenotypic data, and sometimes, with our pedigree data. The success of this depends on an additional set of checks, verifications, filters, and careful preparation of all these datasets in order to be able to fit our genetic models succesfully and to obtain our required output of interest.

As indicated before, `ASRgenomics` aims to assist in the analytical pipelines or workflows required for genomic analyses. These pipelines are specific to each breeding program and they will also depend on the type and quality of the information available. In the following section, we illustrate this flow in a series of steps, with the main goal of implementing genomic prediction.

1. **Preparing molecular matrix $\mathbf{M}$.** Read SNP data and perform quality control and filtering. In some cases, pruning or simple imputations might be required. This step ensures that the data presents reasonable levels of information.

2. **Verifying structure of the population.** Use different tools, with the marker matrix $\mathbf{M}$ or a genomic-based matrix $\mathbf{G}$, to determine the structure of the genotyped individuals. Tools such as clustering and PCA analyses help reveal patterns in the data, allowing us to identify population structure.

3. **Generating initial genomic-based kinship matrix $\mathbf{G}$**. The genomic matrix is generated and used to obtain further insight into the structure of the population or to identify inconsistencies in the marker matrix $\mathbf{M}$.

4. **Verifying genomic matrix $\mathbf{G}$.** Perform validation and checking; for example, by identifying pairs of individuals that appear to be duplicates, or individuals with unreasonable levels of inbreeding.

5. **Correcting and tuning-up the genomic matrix $\mathbf{G}$**. Here, inconsistencies in the matrix $\mathbf{G}$ are dealt with. This might require revisiting the original molecular data or phenotypic records for additional filters or checks. In addition, we can tune up the matrix $\mathbf{G}$ (or any kinship matrix $\mathbf{K}$) to help with its stability in later use.

6. **Generating pedigree-based kinship matrix $\mathbf{A}$.** If pedigree information is available, it can be used to detect further inconsistencies in the matrix $\mathbf{G}$. This can require corrections for the pedigree, the molecular data or both. In addition, the matrices $\mathbf{A}$ and $\mathbf{G}$ might be aligned to obtain a more stable $\mathbf{G}$ matrix.

7. **Generating final genomic matrix G.** After performing a few iterations of cleaning, correcting pedigree, and tuning up and re-assessment of the matrix **G**, a final genomic matrix is generated and it is now available for any of the downstream statistical analyses.

8. **Generating inverse of genomic matrix G.** For some statistical packages (such as `ASReml-R`) the inverse of the **G** matrix is required. This matrix is generated, but tools are used to check if it is ill-conditioned and whether some form of tune-up or correction is required.

9. **Matching a genomic-based or any kinship matrix with phenotypic data.** As we get ready for our statistical analyses, the subset of phenotypic data is obtained for those individuals present in the kinship matrix. In turn, this also allows us to detect further inconsistencies between these datasets.

10. **Fitting linear mixed model.** We proceed to fit our statistical genetics model, assess distributional assumptions and verify the inclusion and significance of model terms.

The above flow is not comprehensive, and might require several other steps (or iterations of these steps) depending on the problem in hand. Also, we have not described the steps following the fitting of the linear mixed model, which might include for example: inspection of residuals for outliers, calculation of heritabilities, extraction of genomic breeding values, estimation of functions of random effects, and in cases of GWAS, extraction and summarization of SNP effects, just to name a few.

3 Reading and Filtering a Molecular Dataset

To illustrate the flow of functions and show some of the capabilities of `ASRgenomics` we are going to use a real dataset from Loblolly Pine (*Pinus taeda* L.) published by Resende et al. (2012). The genotypic portion of this dataset contains a total of 655 genotypes and 4,853 SNPs (coded as 0, 1, and 2, and here -9 is used for missing data). This is a subset of the original dataset where some full-sib genomic records have been artificially eliminated for illustration purposes.

We can call this dataset directly from `ASRgenomics` using:

```
data(geno.pine655)
```

and we can observe the first five genotypes (rows) and first five SNP markers (columns) with:

```
geno.pine655[1:5, 1:5]
```

```
##         X0.10024.01.114 X0.10037.01.257 X0.10040.02.394 X0.10040.02.41 X0.10044.01.392
## 1087120              -9               1              -9             -9              -9
## 1085618               2               2               2              2               2
## 1091040               2               1               2              2               2
## 1091686               2               1               2              2               2
## 1082624               2               2               0              1               1
```

The dataset `geno.pine655` is of class `matrix` and you can see the use of `-9` to indicate missing data, but `NA` or other representations of missing values are also valid within `ASRgenomics`.

A reasonable initial step is to filter this molecular dataset, as it might contain large amounts of missing values and/or non-informative markers. This quality control step is critical to facilitate downstream analyses and further calculations (particularly to ensure that the genomic-based matrix **G** is invertible. For this, the function `qc.filtering()` can be used as illustrated below:

```
M_filter <- qc.filtering(M = geno.pine655, base = FALSE, ref = NULL,
    marker.callrate = 0.2, ind.callrate = 0.2, maf = 0.05, heterozygosity = 0.95,
    Fis = 1, impute = FALSE, na.string = "-9", plots = TRUE)
```

```
## Initial marker matrix M contains 655 individuals and 4853 markers.
```

```
## A total of 119670 values were identified as missing with the string -9 and were
replaced by NA.
```

```
## A total of 156 markers were removed because their proportion of missing values was
equal or larger than 0.2.
```

```
## A total of 11 individuals were removed because their proportion of missing values was
equal or larger than 0.2.

## A total of 1626 markers were removed because their MAF was smaller than 0.05.

## A total of 0 markers were removed because their |F| was larger than 1.

## A total of 25 markers were removed because their heterozygosity was larger than 0.95.

## Final cleaned marker matrix M contains 2.37% of missing SNPs.

## Final cleaned marker matrix M contains 644 individuals and 3046 markers.
```

This function removes markers and individuals according to some specifications. The options considered above are detailed below in the order in which they are implemented in the code; however, note that this filtering process is internally repeated twice as some markers (or individuals) are dropped, affecting the calculations:

- `marker.callrate = 0.2` removes markers with 20% or more missing values.
- `ind.callrate = 0.2` removes individuals with 20% or more missing values.
- `maf = 0.05` removes markers with minor allele frequency (MAF) below 0.05.
- `Fis = 1` removes markers with inbreeding value Fis larger than 1.
- `heterozygosity = 0.95` removes markers with observed heterozygosity larger than 0.95.

Note, we have used the threshold value of 1 for `Fis`, requesting, for this example, not to eliminate any markers under this criteria.

The messages from this function reports that 11 individuals were dropped, and a total of 156 + 1,626 + 25 markers were eliminated. In addition, a new cleaned matrix, `M.clean`, was generated which can be explored with the commands:

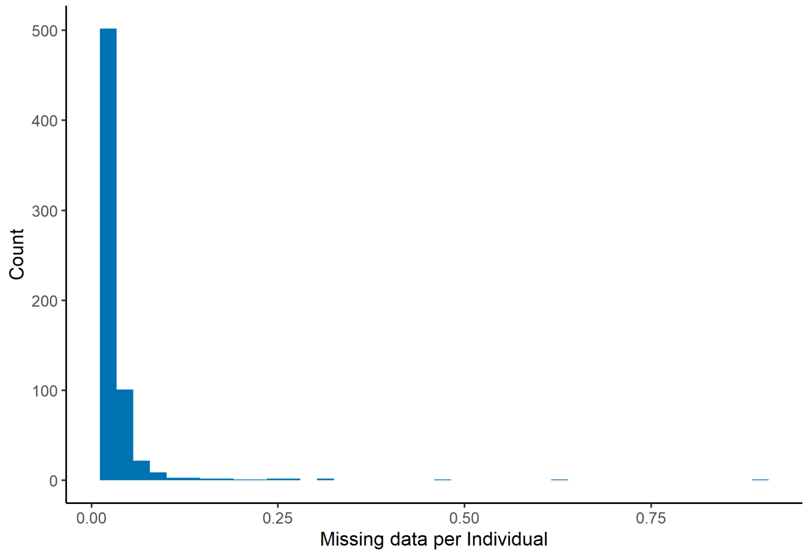
```
M_filter$M.clean[1:5, 1:5]
dim(M_filter$M.clean)
```

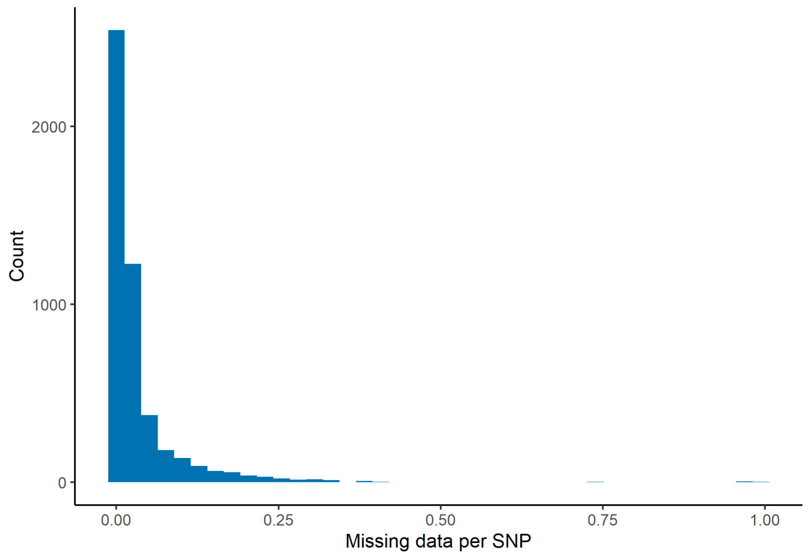
```
##         X0.10037.01.257 X0.10040.02.394 X0.10040.02.41 X0.10044.01.392 X0.10048.01.60
## 1085618               2               2              2               2              1
## 1091040               1               2              2               2              2
## 1091686               1               2              2               2              2
## 1082624               2               0              1               1              0
## 1088628               1               2              2               1              1

## [1]  644 3046
```

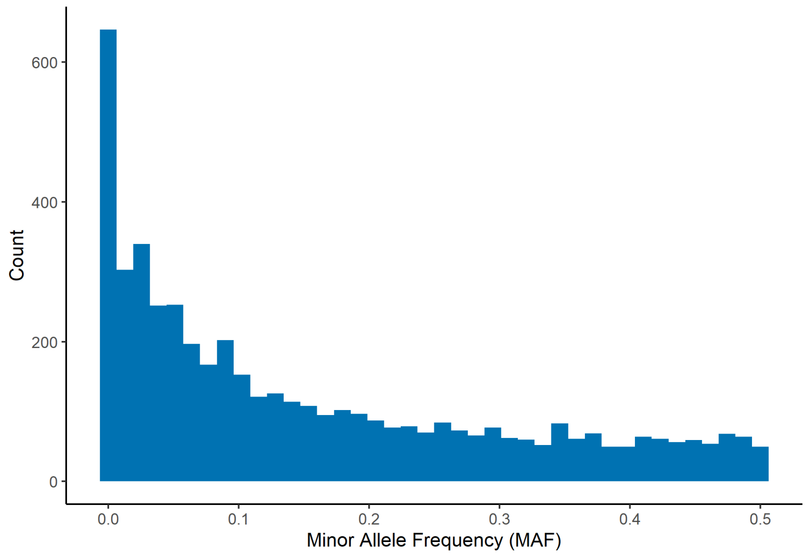
The dimension of this cleaned genomic matrix is 644 individuals by 3,046 markers.

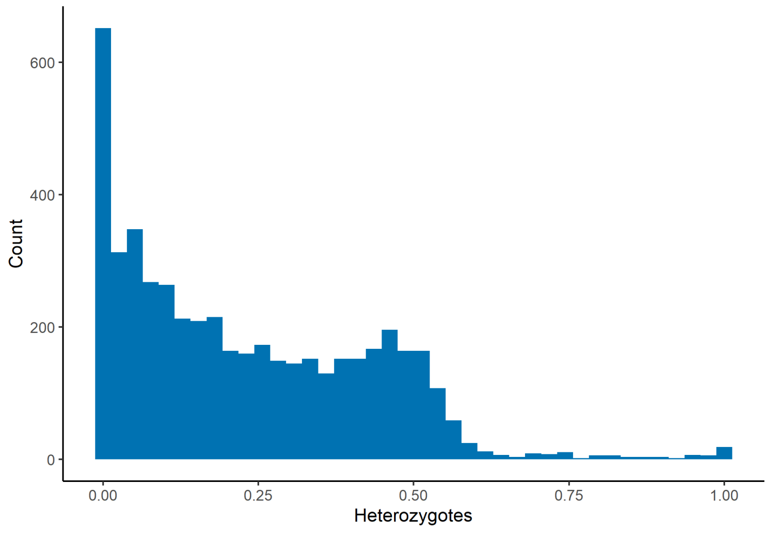
The same function produces five plots based on the original `geno.pine655` matrix, that can be inspected to decide better what threshold level to use on each of the criteria. Note that these plots are obtained using the original data as read in the **M** matrix.

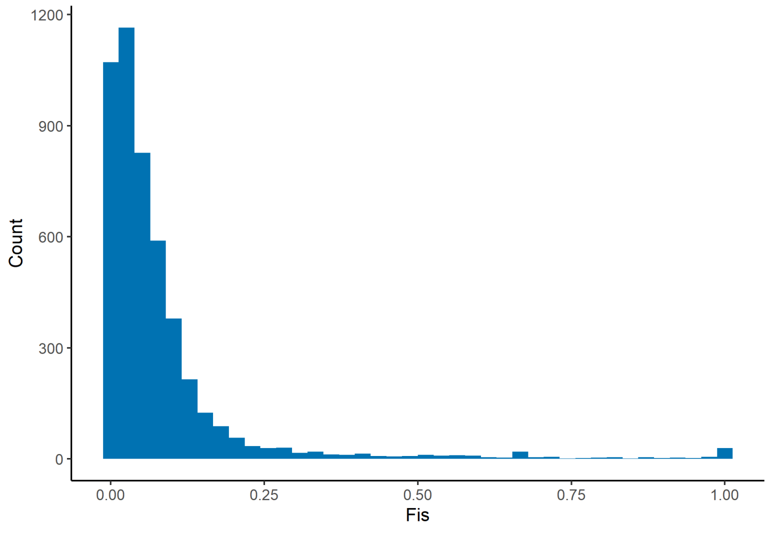
```
M_filter$plot.missing.ind
M_filter$plot.missing.SNP
M_filter$plot.maf
M_filter$plot.heteroz
M_filter$plot.Fis
```






Count
600
400
200
0
0.0
0.1
0.2
0.3
0.4
0.5
Minor Allele Frequency (MAF)


Count
600
400
200
0
0.00
0.25
0.50
0.75
1.00
Heterozygotes


Count
1200
900
600
300
0
0.00
0.25
0.50
0.75
1.00
Fis

Previously in our filtering, we used the option `impute = FALSE`, which has not implemented any form of imputation on the $\mathbf{M}$ matrix. However, it is possible to perform a simple imputation (average method) of missing values within this function. This method is only recommended when the proportion of missing values in the matrix is less than 5% (in this case, after filtering we only have ~2.4% missing values). More sophisticated imputations can be used but these need to be implemented using specific software outside of `ASRgenomics`.

4 Generating a Kinship Matrix

In order to fit a genomic-BLUP (or GBLUP) model for either genomic prediction (GP) or GWAS analyses, you will need to generate a genomic-based kinship matrix **G**. This can be easily obtained with the function `G.matrix()` using the cleaned **M** matrix produced before:

```
G <- G.matrix(M = M_filter$M.clean, method = "VanRaden", na.string = NA)$G
```

The dimension of this matrix is 644 $\times$ 644 individuals, and the first few records are:

```
G[1:5, 1:5]
```

```
##            1085618   1091040    1091686   1082624    1088628
## 1085618  0.948774  0.057050  0.0565112 -0.014318  0.1023679
## 1091040  0.057050  0.931190  0.0741684 -0.021068  0.1623575
## 1091686  0.056511  0.074168  0.9272810 -0.011731  0.0082731
## 1082624 -0.014318 -0.021068 -0.0117306  1.019321 -0.0180782
## 1088628  0.102368  0.162358  0.0082731 -0.018078  0.9702646
```

It is possible to generate the additive relationship matrix (using the methods of `"VanRaden"` or `"Yang"`) and the dominant relationship matrix (methods of `"Su"` or `"Vitizeca"`). In the case of high levels of inbreeding in a population we recommend the use of the `"Yang"` method as it tends to produce more stable matrices. Further details on these methods can be found in Nazarian and Gezan (2016).

The pedigree-based kinship matrix **A** is also very useful for genomic analyses, and we will explore it later. This matrix requires the pedigree file as a data frame. For the Loblolly Pine study this is loaded below and a portion of it is shown:

```
data(ped.pine)
head(ped.pine)
tail(ped.pine)
```

```
##   Indiv Mother Father
## 1 14006      0      0
## 2 14046      0      0
## 3 14060      0      0
## 4 14070      0      0
## 5 14104      0      0
## 6 14106      0      0
```

```
##        Indiv Mother Father
## 2029 1085440 142024  44112
## 2030 1085916  52010  22022
## 2031 1087214  14114  20012
## 2032 1090230  52004  50148
## 2033 1091200  50148  14006
## 2034 1092034  44128  44112
```

The pedigree data frame `ped.pine` has each individual identified together with its maternal and paternal parents. There are a few conditions for this data frame to be valid: the individuals need to be sorted by generation and each individual needs to have their parents specified (with a `0` or `NA` whenever these are unknown).

In order to obtain the pedigree-based **A** matrix we will use the function `Amatrix()` from the library `AGHmatrix` (Amadeu et al. 2016):

```
A <- AGHmatrix::Amatrix(data = ped.pine)
```

This matrix has a larger dimension than our **G** matrix, containing a total of 2,034 individuals. As before, we can explore a few records with:

```
A[601:605, 601:605]
```

```
##         1087776 1087786 1087806 1087808 1087810
## 1087776     1.0     0.5     0.0     0.0     0.0
## 1087786     0.5     1.0     0.0     0.0     0.0
## 1087806     0.0     0.0     1.0     0.5     0.5
## 1087808     0.0     0.0     0.5     1.0     0.5
## 1087810     0.0     0.0     0.5     0.5     1.0
```

Here, we can clearly identify some full-sib individuals (a value of 0.5) and others completely unrelated (a value of 0.0).

5 Diagnostics on the Kinship Matrix

The genomic matrix **G** obtained earlier, and in general any genomic matrix generated with `ASRgenomics` or other packages, will always correspond to an estimation of the true relationships based on the available observed markers. In our Loblolly Pine example, only 3,046 SNPs were used to generate the matrix **G**, and therefore it is prone to sampling errors and other inconsistencies. A small number of markers (*e.g.*, $<$ 1,000 SNPs), poor genotyping quality, high levels of inbreeding and large portions of missing data are likely to affect even more the quality of this matrix. These problems are reduced with adequate filtering, as we illustrated previously, but they might still persist.

`ASRgenomics` incorporates a series of functions that will help us with diagnostics, exploration and further tuning up of kinship matrices in order to reduce inconsistencies. The main function to use is `kinship.diagnostics()` as shown below:

```
check_G <- kinship.diagnostics(K = G, diagonal.thr.small = 0.8,
    diagonal.thr.large = 1.2, duplicate.thr = 0.95)
```

```
## Matrix dimension is: 644x644


## Range diagonal values: 0.84781 to 1.09544


## Mean diagonal values: 0.99592


## Range off-diagonal values: -0.16193 to 0.67935


## Mean off-diagonal values: 0.01379


## There are 0 extreme diagonal values, outside < 0.8 and > 1.2


## There are 0 records of possible duplicates, based on:
k(i,j)/sqrt[k(i,i)*k(j,j)] > 0.95
```

This function generates several diagnostic reports on the matrix, including the range and mean values of the diagonal and off-diagonal values. Here, no observations were reported as extreme on the diagonals (with a *reasonable* range specified between 0.8 and 1.2), and no duplicate genotypes were reported (based on the correlation value between individuals found on the off-diagonal being larger than 0.95). Thresholds for these flagged values can be modified as desired.

The flagged elements of the diagonal, or reported duplicates can be listed with:

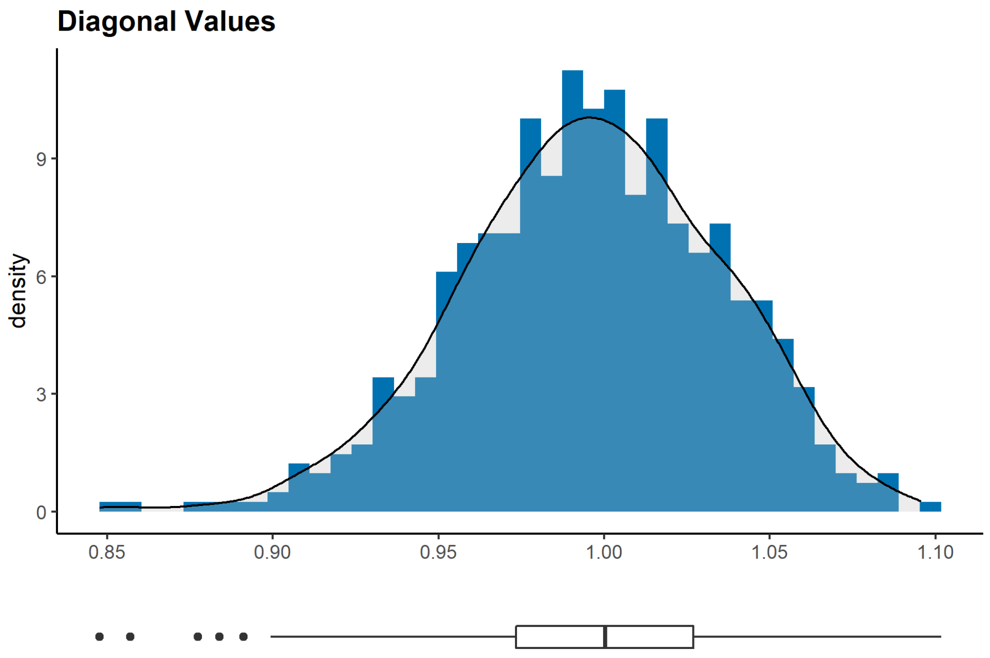
```
check_G$list.diagonal
check_G$list.duplicate
```

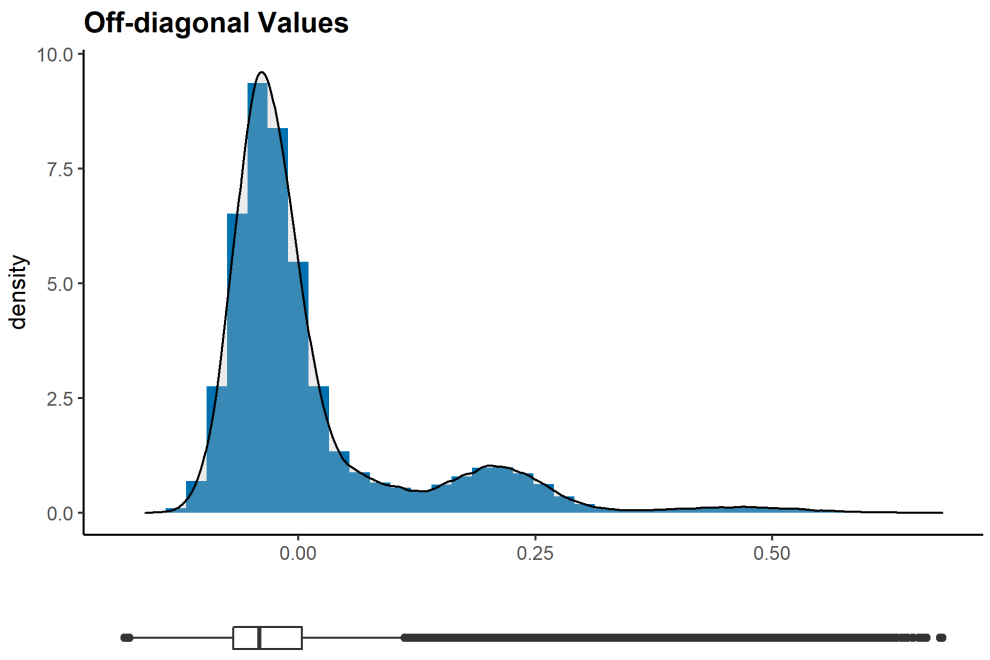
The diagonal values correspond to $1 + F_i$ where F_i is the inbreeding coefficient that can take a theoretical range between 0 and 1. Hence, any values on the diagonal of the **G** matrix that are much lower than 1 and higher than 2 are considered suspicious. However, if the population is known to not have large levels of inbreeding, then these values should not be much larger than 1 or 1.2. Reported duplicate individuals can correspond to those with genotyping issues, and therefore we need to evaluate if they should be dropped from any further analyses.

In addition, you can view histogram plots for the diagonal and off-diagonal values using:

```
check_G$plot.diag
check_G$plot.offdiag
```

The first plot has a reasonable shape, and note that it is centred at ~ 0.995, just slightly lower than what is expected for non-inbred individuals (*i.e.*, 1.0). The second plot, provides several modes corresponding, for this population, to the unrelated individuals, half-sibs and full-sibs, and these modes should be centred at 0, 0.25 and 0.50, respectively. However, this is not happening with our estimated genomic matrix **G**, reflecting bias in the estimation of the genomic relationships between individuals.





`ASRgenomics` has a few additional tools to help identifying issues in this or any other kinship matrices. For example, the function `matchG2A()` can be used to compare the values from the pedigree-based matrix against the genomic-based relationship matrix. This is something we will explore further in the next section.

6 Tuning a Genomic-based Kinship Matrix

Clearly, as we noted in the previous section, our calculated genomic matrix **G** has some bias, with genomic relationships underestimated from what is expected based on pedigree information. Such bias is not uncommon in any genomic matrix, as its calculation requires knowledge of the allele frequencies of the base population (*i.e.*, founders) but this is often not available as we only have the genotyping for a sample of individuals in the population.

Nevertheless, this bias and other inconsistencies can be easily managed within `ASRgenomics` using the `G.tuneup()` function. The main *tuning* options available are: bending, blending and aligning of the **G** matrix.

Given that we have identified overall bias in the relationships, we will proceed to perform an *alignment* of our **G** matrix based on information on the expected relationships as included in the previously calculated **A** matrix from the pedigree.

In order to make these two matrices **G** and **A** compatibles, the alignment implemented in `ASRgenomics` follows the procedure suggested by Christensen et al. (2012), which basically modifies the diagonal and off-diagonal values of a **G** matrix to match the values of the provided **A** matrix for the common genotypes. This concept is implemented by solving the following system of equations for α and β:

$$\alpha + \beta \times mean(diag(\mathbf{G})) = mean(diag(\mathbf{A}))$$
$$\alpha + \beta \times mean(\mathbf{G}) = mean(\mathbf{A})$$

which results in the modified aligned matrix:

$$\mathbf{G_b} = \hat{\alpha} + \hat{\beta} \times \mathbf{G}$$

However, before proceeding, we need the portion of the individuals from the **A** matrix (which in our example has dimensions 2,034 × 2,034) that are present in our **G** matrix (with dimensions 644 × 644). To do this we use the function `matchG2A()`, as shown below:

```
G2A <- match.G2A(A = A, G = G, clean = TRUE, ord = TRUE, mism = TRUE,
    RMdiff = TRUE)
```

```
## All 644 individuals from matrix G match those individuals from matrix A.

## Matrix A has 1390 individuals (out of 2034) NOT present on matrix G.
```

which produces the *matched* matrices `G2A$Aclean` and `G2A$Gclean` that we need, both of dimensions 644 × 644.

Because we used the option `clean = TRUE` we have the matching matrices from above, with the same dimensions and **exactly** the same order of the genotypes. If we had genotypes missing in **G** but present in **A**, these will all be dropped, and vice-versa.

A look at some of the values of these cleaned matrices shows some interesting differences, and the bias is evident:

```
G2A$Aclean[343:347, 343:347]
```

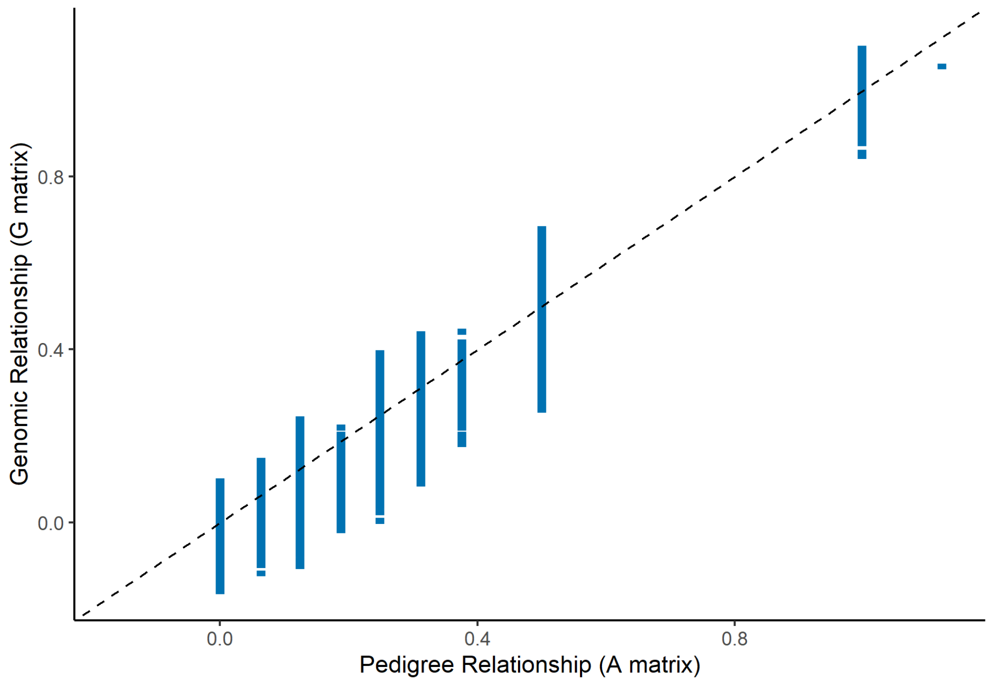
```
##         1087776 1087786 1087806 1087808 1087810
## 1087776     1.0     0.5     0.0     0.0     0.0
## 1087786     0.5     1.0     0.0     0.0     0.0
## 1087806     0.0     0.0     1.0     0.5     0.5
## 1087808     0.0     0.0     0.5     1.0     0.5
## 1087810     0.0     0.0     0.5     0.5     1.0
```

```
G2A$Gclean[343:347, 343:347]
```

```
##           1087776  1087786   1087806  1087808  1087810
## 1087776  0.975246  0.38692 -0.080196 -0.06751 -0.05307
## 1087786  0.386917  1.01028 -0.121400 -0.11889 -0.10270
## 1087806 -0.080196 -0.12140  0.946546  0.45083  0.48740
## 1087808 -0.067510 -0.11889  0.450827  0.95207  0.38793
## 1087810 -0.053070 -0.10270  0.487399  0.38793  0.98878
```

The function `matchG2A()`, as stated earlier, has additional diagnostics that are useful for assessing both of these matrices together. A useful scatterplot that can be generated is one that pairs the values from **G** against **A** from these cleaned matrices.

```
G2A$plotG2A
```



Note that in this plot the values are concentrated on vertical lines for the **A** matrix (as the pedigree is shallow) with reasonable ranges from the **G** matrix. Points that are found outside the expected relationships can be the result of errors in the pedigree or in the molecular data and should be explored carefully. However, in this case, there are no large inconsistencies.

Also, this function produces an output matrix with the pairs (points) shown in the above plot, that if requested (with the option `RMdiff = TRUE`) it can help identify matching errors or inconsistencies between these matrices. The first few records in this example are shown below as part of the data frame `G2A$RM`:

```
head(G2A$RM)
```

```
##   Row Col   IDRow   IDCol AValue  GValue  absdiff Diag
## 1   1   1 1080008 1080008    1.0 1.01582 0.015817    1
## 2   2   1 1080024 1080008    0.5 0.43065 0.069353    0
## 3   2   2 1080024 1080024    1.0 1.05715 0.057150    1
## 4   3   1 1080030 1080008    0.5 0.46777 0.032228    0
## 5   3   2 1080030 1080024    0.5 0.52349 0.023487    0
## 6   3   3 1080030 1080030    1.0 1.06642 0.066421    1
```

This dataset, presented in lower-diagonal form, has the calculated relationships for both of the input matrices together with their absolute difference. In addition, the last column identifies values belonging to the diagonal (1) or off-diagonal (0).

Many additional checks can be performed on this dataset. For example, we can explore the observations that have absolute differences larger than 0.20, and these can be displayed with:

```
head(G2A$RM[G2A$RM$absdiff > 0.2, ])
```

```
##       Row Col   IDRow   IDCol AValue  GValue absdiff Diag
## 2997   77  71 1081474 1081426    0.5 0.29274 0.20726    0
## 3398   82  77 1081598 1081474    0.5 0.29455 0.20545    0
## 20095 200 195 1084642 1084618    0.5 0.26982 0.23018    0
## 21727 208 199 1084732 1084632    0.5 0.25987 0.24013    0
## 21936 209 200 1084734 1084642    0.5 0.26988 0.23012    0
## 29643 243 240 1086246 1086222    0.5 0.29748 0.20252    0
```

This subset corresponds to a total of 648 records, and from the reduced output above we can see that potentially some individuals in the pedigree are declared as full-sibs (`AValue` = 0.50), but they are possible half-sibs (`GValue` $\sim$ 0.25). In addition, we do not show but some individuals were declared as half-sibs (`AValue` = 0.25) when in fact they are unrelated (`GValue` $\sim$ 0). Hence, these results indicate that some pedigree corrections might be required. We recommend a careful assessment, as these differences reflect errors in pedigree or DNA samples, and they are likely to affect downstream analyses.

Finally, we can proceed with our alignment using the clean `Gclean` and `Aclean` matrices from above. This is executed using the function `G.tuneup()` as:

```
G_align <- G.tuneup(G = G2A$Gclean, A = G2A$Aclean, align = TRUE)$Gb
```

```
## Reciprocal conditional number for original matrix is: 0.000212653427326154

## Determinant for original matrix is: 2.76768888610265e-251

## Matrix was ALIGNED.

## Reciprocal conditional number for tune-up matrix is: 0.000196850315087818
```

Some of the reports on this function (presented for blend, bend or align) are in reference to the stability of this matrix, and are associated with the feasibility of the matrix inversion of the generated tuned-up matrix. For the reciprocal conditional number, values near zero are associated with an ill-conditioned matrix (*e.g.*, $<$ 1e-05). For moderate size matrices ($<$ 1,500), the determinant is also calculated, where again values at or near zero are associated with a singular or near-singular matrix, respectively. Further details will be considered later when we obtain the inverse of this matrix with `ASRgenomics`.

Lets explore some of the values of the original (cleaned) matrix and its aligned version:

```
G2A$Gclean[343:347, 343:347]
```

```
##           1087776  1087786   1087806  1087808  1087810
## 1087776  0.975246  0.38692 -0.080196 -0.06751 -0.05307
## 1087786  0.386917  1.01028 -0.121400 -0.11889 -0.10270
## 1087806 -0.080196 -0.12140  0.946546  0.45083  0.48740
## 1087808 -0.067510 -0.11889  0.450827  0.95207  0.38793
## 1087810 -0.053070 -0.10270  0.487399  0.38793  0.98878
```

```
G_align[343:347, 343:347]
```

```
##           1087776   1087786   1087806   1087808   1087810
## 1087776  0.980271  0.413438 -0.036610 -0.024387 -0.010475
## 1087786  0.413438  1.014029 -0.076308 -0.073887 -0.058295
## 1087806 -0.036610 -0.076308  0.952620  0.475012  0.510248
## 1087808 -0.024387 -0.073887  0.475012  0.957938  0.414412
## 1087810 -0.010475 -0.058295  0.510248  0.414412  0.993307
```

As expected, there are differences between these matrices, but the aligned matrix `G_align` is closer to our expected values.

It is recommended to perform another evaluation of the matching of the matrices in order to explore once more the object `RM`, using:

```
Ga2A <- match.G2A(A = A, G = G_align, clean = TRUE, ord = TRUE,
    mism = TRUE, RMdiff = TRUE)
```

```
## All 644 individuals from matrix G match those individuals from matrix A.
```

```
## Matrix A has 1390 individuals (out of 2034) NOT present on matrix G.
```

```
dim(Ga2A$RM[Ga2A$RM$absdiff > 0.2, ])
```

```
## [1] 90  8
```

This time we only find 90 records (instead of the 648 we found earlier) that have an absolute difference larger than 0.20, reflecting a matrix with better quality. And now, we can perform another set of diagnostics on the aligned matrix, as shown in the code below:

```
check_G_align <- kinship.diagnostics(K = G_align)
```

```
## Matrix dimension is: 644x644
```

```
## Range diagonal values: 0.85749 to 1.09607
```

```
## Mean diagonal values: 1.00019
```

```
## Range off-diagonal values: -0.11535 to 0.69519
```

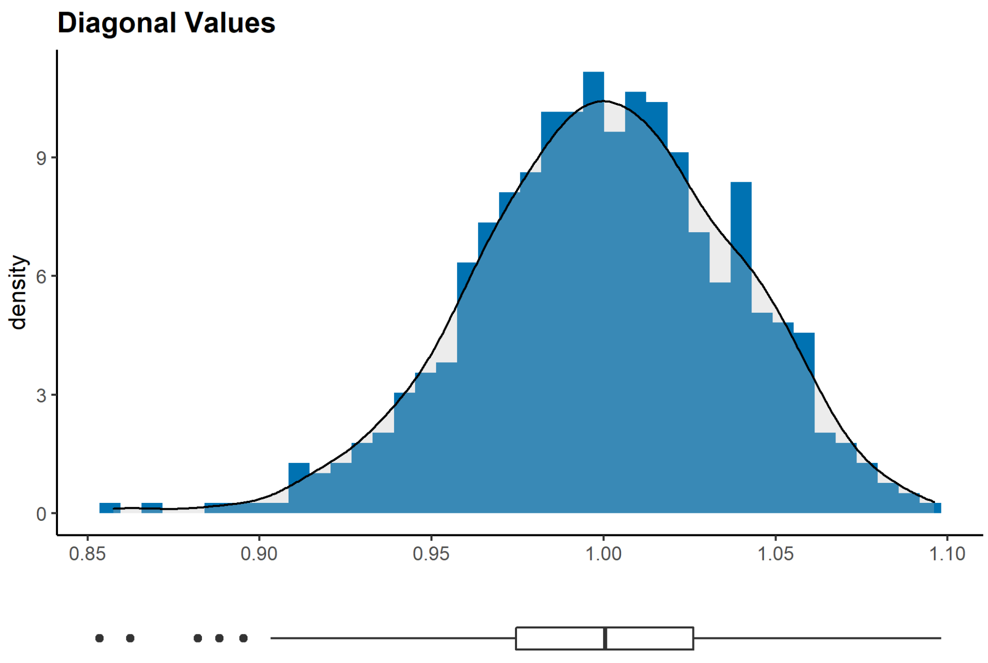
```
## Mean off-diagonal values: 0.05394
```

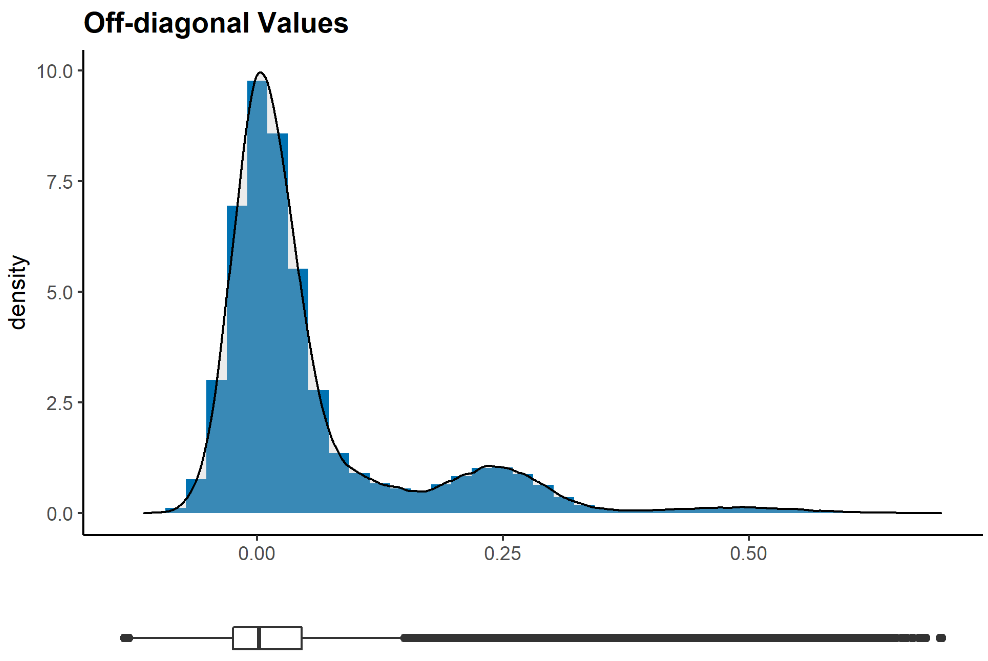
```
## There are 0 extreme diagonal values, outside < 0.8 and > 1.2
```

```
## There are 0 records of possible duplicates, based on:
k(i,j)/sqrt[k(i,i)*k(j,j)] > 0.95
```

Once more, we can request the histograms for the diagonal and off-diagonal values.

```
check_G_align$plot.diag
check_G_align$plot.offdiag
```





The most striking aspect of the off-diagonal plot is that its three modes are now centred at approximately 0, 0.25 and 0.5, as we would expect given the pedigree information. Hence, we will use this matrix for downstream analyses, as it has eliminated that bias.

The function `G.tuneup()` includes two additional methods for tune-up, these are: `blend` and `bend`. Both of these methods, as with the `align` option, help to improve the stability of the inverse (and avoid ill-conditioning).

Under *blending* the **G** matrix is averaged with another matrix, ideally a pedigree-based **A** matrix, but if this is not available or it is unreliable, then an identity matrix of the same dimensions is used. Its general expression is:

$$\mathbf{G_b} = (1 - \texttt{pblend}) \times \mathbf{G} + \texttt{pblend} \times \mathbf{A}$$

where `pblend` is the proportion of the matrix **A** to blend in.

Under *bending* the original **G** matrix is adjusted to obtain a near positive definitive matrix, which is done by making some of its very small or negative eigenvalues slightly positive. This option makes use of the internal R function `Matrix::nearPD()`.

We are not exploring these methods here, but if you require further details we recommend to read Nazarian and Gezan (2016).

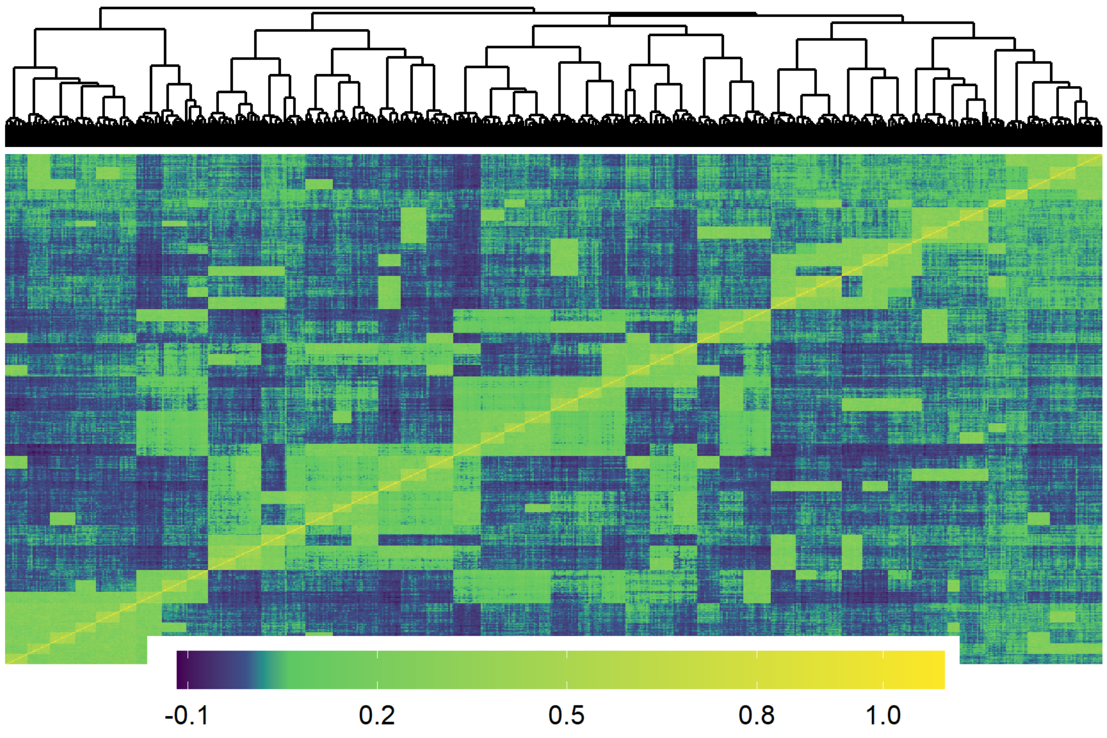
7 Exploring the Kinship Matrix

In this section we will present a few additional functions from `ASRgenomics` that are useful for obtaining greater insight from any kinship matrix **K**. These functions can help us understand the population, generate output for use in other analyses, and also to identify potential issues that might need to checked/corrected.

7.1 Heatmap and Dendrogram from a Kinship Matrix

The function `kinship.heatmap()` is useful for displaying and exploring kinship matrices. This function produces an enhanced heatmap plot based on a provided kinship matrix together with a dendogram. This plot can be used to visualise the structure of a breeding population. For example, we can display the `G_align` matrix with:

```
kinship.heatmap(K = G_align, dendrogram = TRUE, row.label = FALSE,
    col.label = FALSE)
```



In the above figure, we can identify ~ 50 sibships. There are also interesting relationships in the off-diagonal denoting a good connectivity between individuals from different families.

7.2 Population Structure with Genomics Data

The cleaning and processing of the marker information to obtain genomic-based kinship matrices and/or their inverses helps not only with genomic predictions but also with other statistical analyses.

For GWAS analyses, genomic matrices are used to control for family relationships (often known as the **K** matrix). In addition, often a matrix is included to control for population structure (often known as the **Q** matrix). The former matrix, as shown before, can be obtained with `ASRgenomics` using the function `G.matrix()`. The latter, can be generated from either the marker data matrix **M** or the genomic-based kinship matrix **G** with the functions `snp.pca()` or `kinship.pca()`, respectively.

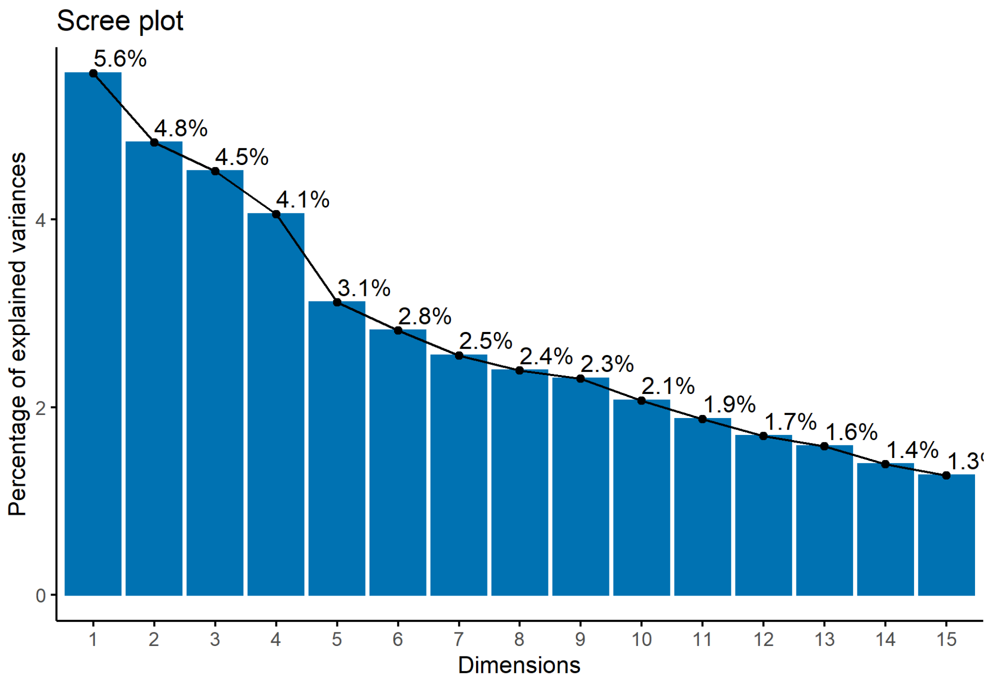
As an example, we illustrate this using the `kinship.pca()` function with the code presented below that makes use of our previously generated `G_align` matrix:

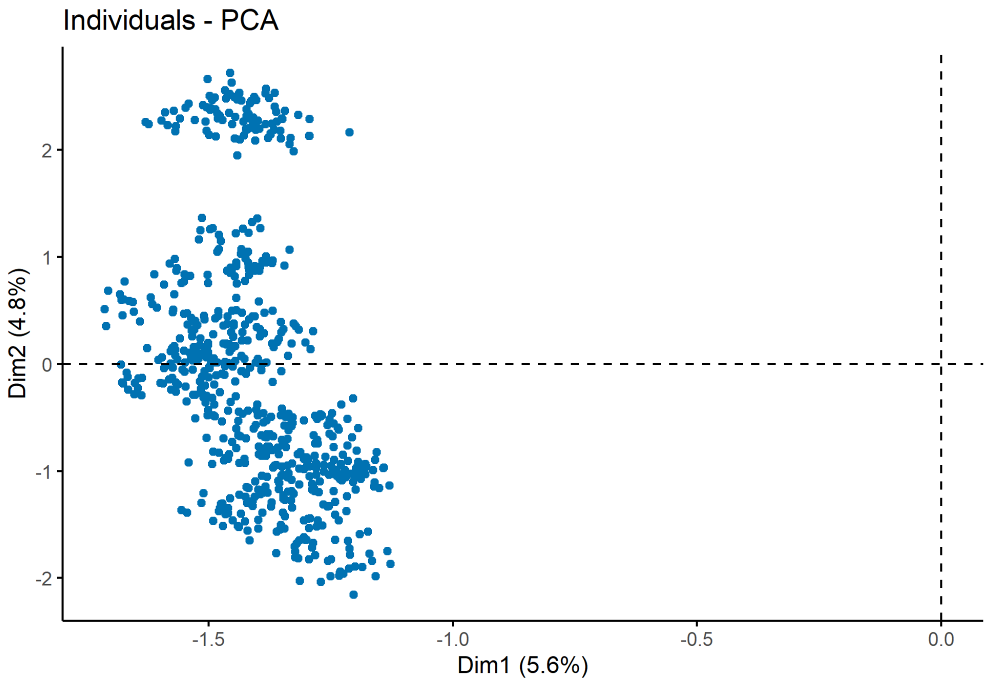
```
pca_pine <- kinship.pca(K = G_align, ncp = 15)
```

Here, we have requested 15 eigenvectors (dimensions) with the option `ncp = 15`, but this function also generates a scree plot and the PCA plot for the first two dimensions. All of this is obtained with the following code:

```
pca_pine$eigenvalues
pca_pine$plot.scree
pca_pine$plot.pca
```

```
##         eigenvalue variance.percent cumulative.variance.percent
## Dim.1      35.7957           5.5583                      5.5583
## Dim.2      31.0358           4.8192                     10.3776
## Dim.3      29.0641           4.5131                     14.8906
## Dim.4      26.1347           4.0582                     18.9488
## Dim.5      20.0666           3.1159                     22.0647
## Dim.6      18.1353           2.8160                     24.8808
## Dim.7      16.4156           2.5490                     27.4298
## Dim.8      15.3856           2.3891                     29.8189
## Dim.9      14.8350           2.3036                     32.1224
## Dim.10     13.3380           2.0711                     34.1936
## Dim.11     12.0630           1.8731                     36.0667
## Dim.12     10.8853           1.6903                     37.7570
## Dim.13     10.1845           1.5814                     39.3384
## Dim.14      8.9513           1.3900                     40.7284
## Dim.15      8.1763           1.2696                     41.9980
```





From the above tables and plots we see that four dimensions explain almost 19% of the variability in the provided genomic matrix and that the vertical grouping observed in the PCA plot are likely to be associated with the different sibships reflecting the structure of this population.

The scores of these 15 dimensions (or PCs) can be accessed using:

```
head(pca_pine$pca.scores)
```

```
##             PC1      PC2     PC3     PC4      PC5      PC6     PC7      PC8
## 1080008 -1.1945 -0.91485 -1.6401 -1.8619 -0.15952 -1.05204 0.46695 -0.37887
## 1080024 -1.2066 -1.02832 -1.6041 -2.0615 -0.29179 -1.04005 0.26372 -0.34721
## 1080030 -1.2105 -1.02527 -1.5416 -2.0125 -0.34073 -1.03975 0.36178 -0.30125
## 1080086 -1.2798 -1.03675 -1.4963 -1.8809 -0.27364 -1.06959 0.40968 -0.38590
## 1080116 -1.2150 -0.90508 -1.6009 -1.9537 -0.28889 -0.79574 0.50727 -0.39928
## 1080132 -1.2014 -1.02834 -1.5310 -1.8730 -0.14265 -0.92580 0.51383 -0.19873
```

```
##             PC9      PC10      PC11     PC12      PC13     PC14     PC15
## 1080008 0.67614  0.050220 -0.072388 0.153265 -0.138734 -0.54841 -0.28107
## 1080024 0.57462 -0.019170  0.072282 0.050428 -0.061157 -0.63495 -0.34329
## 1080030 0.73670 -0.121177  0.102896 0.140770 -0.072745 -0.67924 -0.28590
## 1080086 0.61338 -0.057232 -0.047775 0.150117 -0.128881 -0.69164 -0.25053
## 1080116 0.68346 -0.014549  0.126670 0.108943 -0.091612 -0.56892 -0.27086
## 1080132 0.75156  0.077890 -0.163355 0.201931 -0.205385 -0.53503 -0.25566
```

These scores can be used in downstream GWAS analyses to form the **Q** matrix.

In the PCA scatterplot of the first two-dimensions it is also possible to draw ellipses around pre-defined groups that are requested with `ellipses = TRUE` and the groups are specified with a factor in the option `groups`. For example, we could specify all full-sib families in this factor.

8 Preparing to Fit a Genomic-BLUP (GBLUP) model with ASReml-R

The majority of the previous dataset cleaning, preparation, and exploring is aimed at obtaining a reliable genomic matrix that can be used to fit a linear mixed model (LMM) to the phenotypic data in what is known as GBLUP. In the next example, we will fit a traditional *animal model*, where the pedigree-based kinship matrix **A** is replaced by its genomic counterpart, the **G** matrix. Later, the fitted model can be used to obtain predictions of other genotypes in what is know as Genomic Prediction.

However, in order to fit this LMM we require proper *communication* between our **G** matrix and the phenotypic data. Mainly, all individuals included in the phenotypic dataset need to be included in the **G** matrix to fit the model; however, not all individuals from the **G** matrix need to have phenotypic information, as often these are the target genotypes for genomic predictions.

The phenotypic data that we will use is also extracted from Resende et al. (2012), and this can be loaded and explored with the commands:

```
data(pheno.pine)
head(pheno.pine)
```

```
##   Genotype Mother Father DBH_Adj
## 1  1090230  52004  50148 -5.4394
## 2  1082740 202096  22022 -4.7640
## 3  1086884 222056 202060 -4.6807
## 4  1084222  44126 142170 -4.5188
## 5  1082164  20022  44090 -4.5146
## 6  1086512  44012  17766 -4.3376
```

In this phenotypic dataset the response variable corresponds to the *deregressed estimated breeding values* (or DEBV) for the trait diameter at breast height (DBH) at 6 years of age from trees grown at site Nassau. The dataset contains a total of 861 loblolly pine (*Pinus taeda* L.) individuals. Recall that we only have 644 genotyped individuals in the `G_align` matrix. Hence, our first step is to identify those individuals that are genotyped to subset the phenotypic data. This can be done with the help of `ASRgenomics` by using the function `match.kinship2pheno()` as shown below:

```
pheno.G <- match.kinship2pheno(K = G_align, pheno.data = pheno.pine,
    indiv = "Genotype", clean = FALSE, mism = TRUE)
```

```
## All individuals within the kinship matrix match the phenotyped individuals.
```

```
## Phenotypic data contains 217 individuals that DO NOT match the kinship
matrix individuals.
```

```
## Phenotypic data contains 644 individuals that match the kinship matrix
individuals.
```

The main objective of this function is to identify matches and mismatches of individuals between the kinship matrix, here `G_align`, and the phenotypic dataset, here `pheno.pine`. In our case, we need the matched individuals from the phenotypic data that are listed under the output:

```
pheno.G$matchesP
```

This contains a total of 644 elements. Note, having individuals on the phenotypic data that are not genotyped will lead to errors in the fitting of the genomic model. Hence, the subset of the phenotypic data that we require is:

```
pheno.subset <- pheno.pine[pheno.G$matchesP, ]
```

The `pheno.subset` dataset, as expected, has 644 rows. Here we are using a single record per genotype, but under the framework of the models fitted with `ASReml-R`, it is possible to use multiple measurements in more complex, and often more realistic, linear mixed models.

The function `match.kinship2pheno()` can also provide a **G** with only the subset of individuals/genotypes that match the phenotypic data by using the option `clean = TRUE`. This is particularly useful where a reduced animal model is desired, as we will be able to fit our LMM to **only** those individuals with phenotypic data, and thus reducing the size of the model to fit.

Given that we have our phenotypic dataset ready and our **G** matrix all prepared and checked (`G_align` in this case), we can now proceed to use `ASReml-R` to fit our model. `ASReml-R` can accept both the **G** matrix or its inverse in full or sparse form. However, we strongly recommend to supply the inverse of **G**, *i.e.* $\mathbf{G}^{-1}$, and in sparse form. This is because obtaining the $\mathbf{G}^{-1}$ before model fitting is allowing us to assess its stability and to avoid ill-conditioning. In addition, a sparse form matrix requires less computer resources (*i.e.*, memory) as only the non-zero lower-diagonal of the relevant matrix is stored.

To obtain this, or any other inverse matrices, we use the function `G.inverse()`. For our example the code looks like:

```
Ginv.sparse <- G.inverse(G = G_align, sparseform = TRUE)$Ginv
```

```
## Reciprocal conditional number for original matrix is: 0.000196850315087818


## Reciprocal conditional number for inverted matrix is: 0.000194808044388302


## Inverse of matrix G does not appear to be ill-conditioned.
```

```
head(Ginv.sparse)
```

```
##      Row Col     Value
## [1,]   1   1  3.894719
## [2,]   2   1  0.155490
## [3,]   2   2  3.953677
## [4,]   3   1 -0.048134
## [5,]   3   2 -0.203026
## [6,]   3   3  4.637955
```

From the report we see that the reciprocal conditional number is small, however it is not too small to indicate problems. This is also confirmed by the message telling us that the matrix does not appear to be ill-conditioned. As an empirical rule values larger than 1e-05 are considered reasonable. In addition, we used the option `sparseform = TRUE` to obtain the inverse in sparse form.

If this inverse fails (*e.g.*, due to the matrix being singular or near singular) or if the report indicates that it is ill-conditioned, then a revisit of the genomic matrix might be necessary. This often requires further evaluations ensuring that, for example, duplicate individuals are dropped, and kinship relationships between individuals are within reasonable ranges, amongst other things including additional tune up.

Critically, the function `G.inverse()` provides the generated matrix $\mathbf{G}^{-1}$ in full or sparse form, with all the required attributes to be used directly in `ASReml-R` without further manipulations. Here, the key elements are the attributes of `"rowNames"`, `"colNames"` and `"INVERSE"` as shown below for our sparse form matrix:

```
head(attr(Ginv.sparse, "rowNames"))
```

```
## [1] "1080008" "1080024" "1080030" "1080086" "1080116" "1080132"
```

```
head(attr(Ginv.sparse, "colNames"))
```

```
## [1] "1080008" "1080024" "1080030" "1080086" "1080116" "1080132"
```

```
attr(Ginv.sparse, "INVERSE")
```

```
## [1] TRUE
```

9 Fitting a Genomic-BLUP (GBLUP) model with ASReml-R

Using ASReml-R we can now proceed to fit our LMM based on an animal model to the response variable DBH_Adj. But first, we will load the asreml library and define factor terms using:

```
library(asreml)
pheno.subset$Genotype <- as.factor(pheno.subset$Genotype)
```

The code to fit our LMM using the asreml() function is:

```
GBLUP <- asreml(fixed = DBH_Adj ~ 1, random = ~vm(Genotype, Ginv.sparse),
    residual = ~idv(units), na.action = na.method(y = "include"),
    workspace = 1e+07, data = pheno.subset)
```

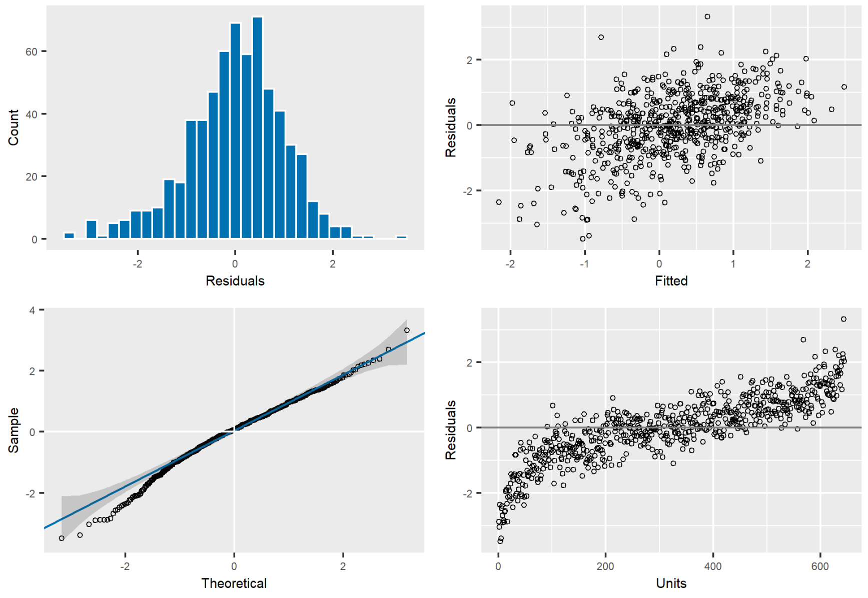
We are not going to explain this code in much detail, as this information is available directly from the ASReml-R manual. However, in the above model we only have a single factor (Genotype) that is assumed to be random, and with the use of the command vm() we assign a variance model that links this factor to the provided inverse Ginv.sparse. As everything in this matrix is in order, this model should not have fitting issues. However, if your $\mathbf{G}^{-1}$ is large ($>$ 2,000 individuals) and/or dense, this model might take some time to fit. You may also need to increase the memory workspace required by the core routines using the workspace option. For additional tricks we recommend you read https://vsni.co.uk/blogs/faster_GBLUP_ASReml-R.

In the above model we have not included other fixed or random effects (such as blocks or plots), but this is easily done. Also, more complex LMMs such as multi-trait or multi-environment trial analyses can be fitted. Again, the software manual is a good resource for these and other statistical analyses.

For our fitted model, we can print out the estimated REML variance components and assess the residual plots by using:

```
summary(GBLUP)$varcomp
plot(GBLUP)
```

```
##                           component std.error z.ratio bound %ch
## vm(Genotype, Ginv.sparse)    1.1717   0.25001  4.6865     P   0
## units!units                  1.3835   0.13688 10.1077     P   0
## units!R                      1.0000        NA      NA     F   0
```



We can also calculate a genomic narrow-sense heritability from this analysis using:

```
vpredict(GBLUP, h2 ~ V1/(V1 + V2))
```

```
##    Estimate       SE
## h2  0.45855 0.071426
```

This is a reasonable heritability for this trait, and the approximated standard error is small.

Finally, the first few genomic expected breeding values (GEBVs) from this model can be extracted with:

```
head(summary(GBLUP, coef = TRUE)$coef.random)
```

```
##                                      solution std.error    z.ratio
## vm(Genotype, Ginv.sparse)_1080008  0.7154172   0.67482  1.0601538
## vm(Genotype, Ginv.sparse)_1080024 -0.4899763   0.67809 -0.7225835
## vm(Genotype, Ginv.sparse)_1080030 -0.0028485   0.66397 -0.0042901
## vm(Genotype, Ginv.sparse)_1080086  0.3045619   0.65774  0.4630444
## vm(Genotype, Ginv.sparse)_1080116  0.0099787   0.67516  0.0147799
## vm(Genotype, Ginv.sparse)_1080132  0.9060461   0.68281  1.3269385
```

It is also possible to use **G** matrices with slightly different tune up options (*e.g.*, blend with different proportions), and this might yield slightly different model fits. We recommend the use of the log-likelihood value to compare models as some fits might be better or more stable than others.

10 Generating a Hybrid Genomic Matrix (H)

In the previous section we successfully fitted a GBLUP genomic prediction model with `ASReml-R` using the inverse of an aligned **G** matrix resulting in a reasonable heritability and estimates of GEBVs that allow us to select outstanding genotypes for a breeding program.

For illustration, in our example, the `G_align` (or corresponding `Ginv.sparse`) matrix only considered the phenotypic and genomic data from the subset of 644 genotyped individuals, thereby limiting our selection to this reduced set. However, we had a phenotypic dataset containing 861 individuals, of which 217 were (artificially) dropped because of their lack of genomic data. This is clearly suboptimal, as all this phenotypic information should be considered in our analyses, and some of these records are from relatives of our genotyped individuals.

In addition, we had available a pedigree dataset for a total of 2,034 individuals, which was also not considered in our GBLUP model. However, we might be interested in obtaining breeding values for the totality of these individuals regardless of the type of information available.

The methodology of single-step GBLUP, or ssGBLUP, allows to combine the information from the pedigree-based kinship matrix **A** together with the genomic-based **G** matrix to generate a so-called hybrid matrix **H**, and uses this matrix (or its inverse) in place of the **G** matrix (or its inverse). This **H** matrix has the same dimensions as the **A** matrix, and for the genotyped individuals, it can use either of their available relationship calculations, or a combination of these.

`ASRgenomics` includes the function `H.inverse()` to facilitate the generation of the inverse of this hybrid matrix, $\mathbf{H}^{-1}$, which can be used directly within `ASReml-R` to fit what is sometimes known as HBLUP. This function can accept as input the **A** matrix, or its inverse $\mathbf{A}^{-1}$, and a previously generated genomic inverse $\mathbf{G}^{-1}$. Further information on these procedures are available in Christensen and Lund (2010) and Legarra et al. (2009).

The function `H.inverse()` contains a few scaling factors to help with the calculation of this inverse and to allow further exploration of the combination of the information from the $\mathbf{A}^{-1}$ and $\mathbf{G}^{-1}$. We follow the specifications described by Martini et al. (2018), which is done by specifying the parameters λ, or the pair τ and ω. The general expression used is:

$$\mathbf{H}^{-1} = \mathbf{A}^{-1} + \begin{bmatrix} 0 & 0 \\ 0 & (\tau \mathbf{G}^{-1} - \omega \mathbf{A_{22}}^{-1}) \end{bmatrix}$$

where $\mathbf{A_{22}}^{-1}$ is the pedigree relationship of the genotyped individuals.

A more common representation of the above expression is found when $\tau = \omega = \lambda$, as shown below:

$$\mathbf{H}^{-1} = \mathbf{A}^{-1} + \begin{bmatrix} 0 & 0 \\ 0 & \lambda(\mathbf{G}^{-1} - \mathbf{A_{22}}^{-1}) \end{bmatrix}$$

To illustrate the `H.inverse()` function we will continue using the Loblolly pine dataset presented earlier, and we supply our **A** matrix and the $\mathbf{G}^{-1}$ obtained from our `G_align` matrix. Note that both of these matrices need to be in full form.

```
Ginv <- G.inverse(G = G_align, sparseform = FALSE)$Ginv
Ginv[1:5, 1:5]
```

```
##           1080008   1080024   1080030   1080086   1080116
## 1080008  3.894719  0.155490 -0.048134 -0.248860  0.245216
## 1080024  0.155490  3.953677 -0.203026  0.021871  0.182718
## 1080030 -0.048134 -0.203026  4.637955 -0.204719 -0.099534
## 1080086 -0.248860  0.021871 -0.204719  4.571356 -0.307563
## 1080116  0.245216  0.182718 -0.099534 -0.307563  4.143026
```

To get the $\mathbf{H}^{-1}$ matrix (in sparse form), which is more computationally efficient than getting the **H** matrix, we will use the following code:

```
Hinv.sparse <- H.inverse(A = A, G = Ginv, lambda = 0.9, sparseform = TRUE)
```

```
## A lambda value was provided and it will be used instead of tau and omega.
```

```
## Matrix A or Ainv has 1390 individuals that are not present in Ginv.
```

This matrix has, as expected, a dimension of 2,034 × 2,034 individuals. In this example, we have used a $\lambda = 0.90$ for the scaling parameter, but other values are possible. Here, a value of 0 indicates that, for those individuals that are genotyped, no information is used from the **G** matrix, and a value of 1 indicates that no information from the **A** matrix is used.

As indicated before, the `H.inverse()` function can accept a different set of scaling parameters, such as τ and ω, as described by Martini et al. (2018), with a range of possible values. We will not explore these options any further but we refer to the above manuscript.

Sometimes, it might be of interest to obtain the **H** matrix directly, and for this we can use the function `H.matrix()` within `ASRgenomics`. This might be computationally intensive, but it can be very useful to explore the hybrid genetic relationship between some of the genotypes, particularly those not genotyped. This **H** matrix is obtained with the code:

```
H <- H.matrix(A = A, Ginv = Ginv, lambda = 0.9, sparseform = FALSE)
```

We can now explore some of the differences between the **A** and **H** matrices to shed light on the relationships between some of the base parents (founders). For example, we have:

```
A[34:39, 34:39]
```

```
##       44126 44128 50148 50152 51810 52004
## 44126     1     0     0     0     0     0
## 44128     0     1     0     0     0     0
## 50148     0     0     1     0     0     0
## 50152     0     0     0     1     0     0
## 51810     0     0     0     0     1     0
## 52004     0     0     0     0     0     1
```

```
H[34:39, 34:39]
```

```
##          44126    44128    50148    50152 51810    52004
## 44126 0.902037 0.059341 0.031676 0.069001     0 0.073155
## 44128 0.059341 0.946418 0.045060 0.032968     0 0.037496
## 50148 0.031676 0.045060 0.947384 0.067636     0 0.055413
## 50152 0.069001 0.032968 0.067636 0.972216     0 0.047243
## 51810 0.000000 0.000000 0.000000 0.000000     1 0.000000
## 52004 0.073155 0.037496 0.055413 0.047243     0 1.035070
```

and some of the offspring:

```
A[601:605, 601:605]
```

```
##         1087776 1087786 1087806 1087808 1087810
## 1087776     1.0     0.5     0.0     0.0     0.0
## 1087786     0.5     1.0     0.0     0.0     0.0
## 1087806     0.0     0.0     1.0     0.5     0.5
## 1087808     0.0     0.0     0.5     1.0     0.5
## 1087810     0.0     0.0     0.5     0.5     1.0
```

```
H[601:605, 601:605]
```

```
##            1087776   1087786   1087806   1087808    1087810
## 1087776  0.9129987  0.392395 -0.027461 -0.016209 -0.0057651
## 1087786  0.3923949  0.925100 -0.056083 -0.048156 -0.0407531
## 1087806 -0.0274612 -0.056083  0.909735  0.453037  0.4857556
## 1087808 -0.0162089 -0.048156  0.453037  0.905317  0.3985635
## 1087810 -0.0057651 -0.040753  0.485756  0.398564  0.9298915
```

As with any other kinship matrix, we can assess the quality of the **H** matrix using the diagnostic tools within `ASRgenomics` as shown below:

```
check_H <- kinship.diagnostics(K = H)
```

We do not show the results, but there were no extreme values reported here, and the plots obtained from this function using this **H** matrix are somewhat different to the ones obtained using the **A** matrix. But given the combination of different sources of information, they still reflect a reasonable kinship matrix.

11 Fitting a Single-Step Genomic-BLUP model (ssGBLUP) with ASReml-R

As we did earlier with GBLUP, we can now proceed to fit our LMM using the $\mathbf{H}^{-1}$ matrix within `ASReml-R`. We will be using similar code as in the previous case, but this time we will use the complete phenotypic data `pheno.pine` that includes a total of 861 individuals, together with the `Hinv.sparse` matrix that includes 2,034 genotypes. This matrix was generated with all the required attributes to be used directly within `ASReml-R`.

We obtained our $\mathbf{H}^{-1}$ previously, however, in the following code we are requesting this matrix in sparse form, and we are defining our relevant factors for our LMM:

```
Hinv.sparse <- H.inverse(A = A, G = Ginv, lambda = 0.9, sparseform = TRUE)
pheno.pine$Genotype <- as.factor(pheno.pine$Genotype)
```

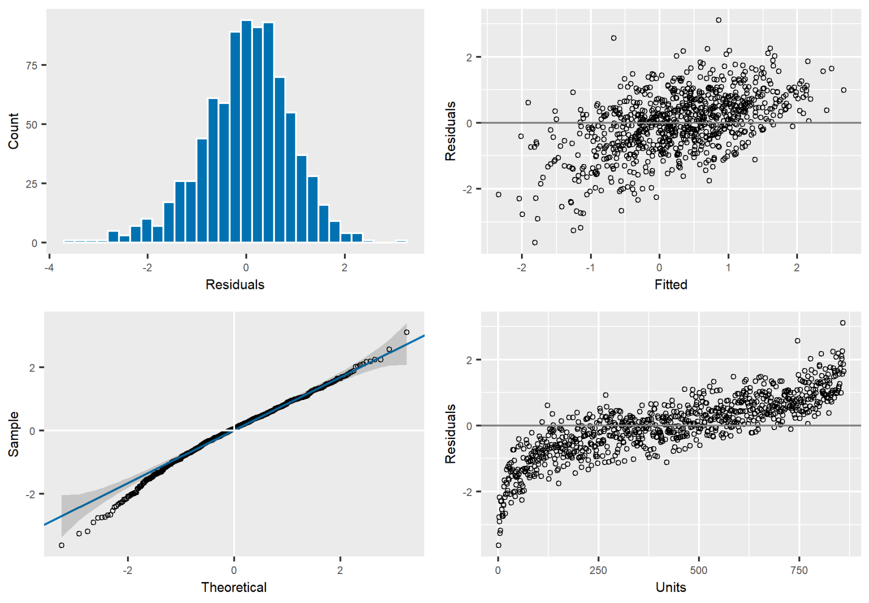
We can then proceed to fit our ssGBLUP model to the response variable `DBH_Adj` using:

```
ssGBLUP <- asreml(fixed = DBH_Adj ~ 1, random = ~vm(Genotype,
    Hinv.sparse), residual = ~idv(units),
    na.action = na.method(y = "include"),
    workspace = 1e+07, data = pheno.pine)
```

As before, the critical element is the use of `Hinv.sparse` associated with the factor `Genotype`, but essentially this is the same model as before.

It is possible to assess the residuals and estimated variance components with:

```
plot(ssGBLUP)
summary(ssGBLUP)$varcomp
```



```
##                              component std.error z.ratio bound %ch
## vm(Genotype, Hinv.sparse)       1.3411   0.24874  5.3916     P   0
## units!units                     1.2447   0.12663  9.8297     P   0
## units!R                         1.0000        NA      NA     F   0
```

These components can be used to calculate a genomic heritability based on our hybrid $\mathbf{H}^{-1}$ matrix as:

```
vpredict(ssGBLUP, h2 ~ V1/(V1 + V2))
```

```
##    Estimate       SE
## h2  0.51864 0.067075
```

This, as before, is a reasonable heritability for this trait that in contrast with our GBLUP analyses has a larger value and slightly smaller approximated standard error.

Finally, we can extract GEBVs from the ssGLUP model. These are shown below for a subset of genotypes together with, for comparison, the ones from our previously fitted GBLUP model:

```
summary(ssGBLUP, coef = TRUE)$coef.random[601:605, ]
```

```
##                                    solution std.error    z.ratio
## vm(Genotype, Hinv.sparse)_1087776 5.7614e-01   0.66289 8.6914e-01
## vm(Genotype, Hinv.sparse)_1087786 7.0410e-01   0.65578 1.0737e+00
## vm(Genotype, Hinv.sparse)_1087806 3.6868e-02   0.67098 5.4946e-02
## vm(Genotype, Hinv.sparse)_1087808 1.0793e-05   0.66676 1.6188e-05
## vm(Genotype, Hinv.sparse)_1087810 2.9264e-01   0.66443 4.4044e-01
```

```
summary(GBLUP, coef = TRUE)$coef.random[343:347, ]
```

```
##                                   solution std.error   z.ratio
## vm(Genotype, Ginv.sparse)_1087776  0.409274   0.66610  0.614437
## vm(Genotype, Ginv.sparse)_1087786  0.512070   0.65973  0.776182
## vm(Genotype, Ginv.sparse)_1087806 -0.054346   0.67207 -0.080864
## vm(Genotype, Ginv.sparse)_1087808 -0.083940   0.66986 -0.125310
## vm(Genotype, Ginv.sparse)_1087810  0.083121   0.67003  0.124056
```

Note that there are some changes in these two sets of solutions, but also that there is a small reduction in their standard errors with the use of the $\mathbf{H}$ matrix. This is likely a result of the increased heritability value, but is also due to the use of the additional information available for each genotype, which was incorporated in the generation of the $\mathbf{H}^{-1}$ to estimate breeding values.

12 Additional Tools in ASRgenomics

In this section we will present a few extra functions from `ASRgenomics` that are useful for additional file reading formats and manipulations of the molecular matrix **M** and also to expand this matrix to generate *synthetic* offsprings.

12.1 SNP Recoding

It is common to have the original molecular matrix **M** available in the bi-allele SNP data format (AA, AG, GG, CC, etc.). This matrix will need to be recoded into the numeric values 0, 1, 2 for use in `ASRgenomics` and other packages. For this, the function `snp.recode()` can be used. In earlier versions of `ASRgenomics` this capability was available under the function `qc.filtering()`.

In the following example, we illustrate its use based on an hypothetical bi-allele matrix named `Mnb`.

```
M.recoded <- snp.recode(M = Mnb, map = mapnb, marker = "marker",
    ref = "ref", alt = "alt", rename.markers = TRUE, na.string = NA)
```

The data frame `mapnp` is optional but in this case was included as it has the information about the reference allele used to calculate the allele proportions. This might be relevant if we want to match the same reference and alternative alleles from other previously available **M** matrices. Here, the options `marker = "marker"`, `alt = "alt"` and `ref = "ref"` indicate the relevant names of the columns within the data frame `mapnb`. Further details can be found in the help associated with this function.

12.2 SNP Pruning

Good filtering of the **M** matrix is recommended in order to avoid issues with the generation of the genomic matrix **G** but also for downstream statistical analyses. Several options for filtering were presented before, but `ASRgenomics` has another important filtering option to consider: SNP pruning.

The function `snp.pruning()` finds and drops some of the SNPs that are highly correlated. This is recommended as a large portion of SNPs in high linkage disequilibrium (LD) can affect genomic selection or GWAS analyses. This function uses the Pearson’s Correlation between the markers (as a *proxy* for LD) and its use is illustrated below:

```
Mpr <- snp.pruning(M = M_filter$M.clean, pruning.thr = 0.9, window.n = 40,
    overlap.n = 10, seed = 1208)
```

```
##
## Creating dummy map.
```

```
##
## Initiating pruning procedure.

## Initial marker matrix M contains 644 individuals and 3046 markers.

## Requesting pruning without chromosome indexing.

##   Iteration: 1

##   Iteration: 2

##   Iteration: 3

##
## Summary statistics.

## Final pruned marker matrix M contains 644 individuals and 2951 markers.

## A total of 95 markers were pruned.

## Range of minor allele frequency after pruning: 0.05 ~ 0.5

## Range of marker call rate after pruning: 80.28 ~ 100

## Range of individual call rate after pruning: 81.67 ~ 99.19
```

In this particular case, some of the markers that have a correlation of 0.90 or higher are eliminated. This is done by blocks (windows), where we have indicated that we want windows of 40 SNPs with an overlap of 10 markers. If a map is provided, then this can be done by chromosome. The report from this function indicates that a total of 95 markers were pruned out, and therefore our cleaned matrix is reduced from 3,046 to 2,951 SNPs.

12.3 Generation of Synthetic Offspring

Sometimes is required to generate molecular data from hypothetical crosses based only on genomic information from the parents. The function `synthetic.offspring()` can be used for this purpose, which will generate genomic data for offspring without marker information.

This is required for some plant breeding programs that deal with crossing of parental inbred lines, where it is possible to generate (*i.e.*, impute) the molecular matrix of any offspring based on this original **M** matrix of the parents. Further details of this function, together with an example, and a description of how heterozygotic records are treated can be found in the associated help.

12.4 Other Tools

Other functions are available within `ASRgenomics`, including `G.predict()` to generate conditional predictions of random variables. There are also additional functions to manipulate and transform matrices, such as `full2sparse()` and `sparse2full()` used to change matrices from full to sparse form and vice versa, respectively.

Further details on these and other functions can be found in the documentation for this package.

13 Closing Remarks

The array of functions implemented in the `ASRgenomics` library are critical tools to facilitate preparation and manipulation of genomic data and to obtain kinship matrices to use in downstream analyses such as genomic prediction and GWAS.

`ASRgenomics` reflects the current advancement and methodologies reported in the available scientific literature to deal with these matrices, including aspects such as manipulations and their tuning up as shown earlier in this manual. A full understanding of the correct procedures to use and to manipulate these matrices requires empirical work using the specific datasets from an operational breeding program or a research project, but we believe `ASRgenomics` should make these decisions easier, faster and more reliable.

This package is in no way comprehensive, and given the fast changing field of genomics and bioinformatics, we suspect additional options and functions will be required in the future. Hence, we consider this library as a dynamic source that in future versions will be expanded to incorporate new functionalities.

Bibliography

Amadeu, R. R., C. Cellon, J. W. Olmstead, A. A. Garcia, M. F. Resende Jr, and P. R. Muñoz. 2016. AGHmatrix: R package to construct relationship matrices for autotetraploid and diploid species: A blueberry example. *The Plant Genome*. 9(3):1–10.

Butler, D., B. R. Cullis, A. Gilmour, and B. Gogel. 2009. ASReml-R reference manual. *The State of Queensland, Department of Primary Industries and Fisheries, Brisbane*.

Christensen, O. F., and M. S. Lund. 2010. Genomic prediction matrix when some animals are not genotyped. P. 1–8 in *Genetics Selection Evolution*,.

Christensen, O., P. Madsen, B. Nielsen, T. Ostersen, and G. Su. 2012. Single-step methods for genomic evaluation in pigs. *Animal*. 6(10):1565–1571.

Legarra, A., I. Aguilar, and I. Misztal. 2009. A relationship matrix including full pedigree and genomic information. *Journal of Dairy Science*. 92(9):4656–4663.

Martini, J. W., M. F. Schrauf, C. A. Garcia-Baccino, E. C. Pimentel, S. Munilla, A. Rogberg-Muñoz, R. J. Cantet, et al. 2018. The effect of the H^{-1} scaling factors τ and ω on the structure of H in the single-step procedure. *Genetics Selection Evolution*. 50(1):1–9.

Nazarian, A., and S. A. Gezan. 2016. GenoMatrix: A software package for pedigree-based and genomic prediction analyses on complex traits. *Journal of Heredity*. 107(4):372–379.

R Core Team. 2020. *R: A language and environment for statistical computing*. R Foundation for Statistical Computing, Vienna, Austria. Available online at: https://www.R-project.org/.

Resende, M., P. Muñoz, M. D. Resende, D. J. Garrick, R. L. Fernando, J. M. Davis, E. J. Jokela, T. A. Martin, G. F. Peter, and M. Kirst. 2012. Accuracy of genomic selection methods in a standard data set of loblolly pine (*Pinus taeda* L.). *Genetics*. 190(4):1503–1510.