User's Manual for ASRgenomics v. 1.0.0

An R package with complementary genomic functions

Prepared by

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

March 22, 2021

User's Manual for ASRgenomics v. 1.0.0

Bibliographical reference

Gezan, S.A., de Oliveira, A.A., and Murray, D. 2021. ASRgenomics: An R package with Complementary Genomic Functions. Version 1.0.0 VSN International, Hemel Hempstead, United Kingdom.

Authors' email addresses

Salvador A. Gezan salvador.gezan@vsni.co.uk
Amanda A. de Oliveira amanda.aoliveira@vsni.co.uk
Darren Murray darren.murray@vsni.co.uk

Companion resources

This R library and associated documentation can be downloaded from:
https://asreml.kb.vsni.co.uk/knowledge-base/asrgenomics/

Acknowledgements

The authors greatly acknowledge the team of testers and evaluators from our circle. They helped to make this tool more robust and useful.

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) and/or Genome-Wide Association Studies (GWAS).

The main tasks considered are:

- Preparing/exploring pedigree/phenotypic data.
- Preparing/exploring genomic data.
- Complementing genomic breeding values.

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 population structure; calculating the genomic matrix **G** and its inverse and assessing its quality; matching pedigree-against genomic-based matrices; tuning up a genomic matrix (bend, blend or align); obtaining the hybrid matrix **H** as required for single-step GBLUP (ssGBLUP).

The routines considered 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) that 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, such as ssGBLUP.

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 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.

In this manual, we will present some of the structure of `ASRgenomics` illustrating its use with examples using datasets provided in this package. `ASRgenomics` is a tool that is provided "as is." We have made 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://asreml.kb.vsni.co.uk/knowledge-base/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`
- `factoextra`
- `ggplot2`
- `ggpubr`
- `reshape2`
- `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:

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

For Linux:

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

For Mac:

```
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>/ASRgenomics1.0.0.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 `? 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 rather complicated. Traditional genetic analyses using only pedigree data requires that the phenotypic data and the pedigree data 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 information, and often, with our pedigree data. The success of this depends on an additional set of checks, verifications, and careful preparation of all these datasets in order to be able to fit our genetic models and to obtain our required output of interest.

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

1. **Preparing molecular matrix M.** Read SNP data and perform quality control and filtering. In some cases, imputation might be performed. This step ensures that data presents reasonable levels of information.
2. **Verifying structure of the population.** Use tools to determine the structure of the genotyped individuals with the matrix **M**. Aspects such as clustering and PCA analyses will allow us to identify some form of structure, and this will help us to see patterns in the data.
3. **Generating first genomic-based kinship matrix G**. The genomic matrix is obtained, and it is used to obtain further insight on the structure of the population or to identify inconsistencies.
4. **Verifying genomic matrix G.** Perform validation and checking; for example, by identifying duplicates, individuals with unreasonable levels of inbreeding, or pairs of individuals that appear to be duplicates.
5. **Correcting and tuning-up genomic-based matrix G**. Inconsistencies are dealt with, and this might require us to go to the original molecular data or phenotypic records. In here, we can also tune-up the matrix **G** to help with its stability for further use.
6. **Generating pedigree-based kinship matrix A.** If pedigree information is available, it can be used to detect further inconsistencies in the matrix **G**. This can require corrections for either the pedigree or the molecular data. In addition, the matrices **A** and **G** might be aligned to obtain a more stable matrix **G**.
7. **Generating final genomic-based matrix G.** After performing a few iterations of cleaning, correcting pedigree, 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-based matrix G.** For some statistical packages (such as ASReml) the inverse of the **G** matrix is required. This matrix is generated, but tools are used to evaluate if it is ill-conditioned and if some form of tune-up or correction is required.
9. **Matching G 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 matrix **G** to ensure that our model fits. 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, and, as with any other model fitting process, we inspect residuals for outliers, calculate heritabilities and verify the inclusion and significance of model terms.

The above flow is not comprehensive, and there might be several other steps depending on the problem in hand. Also, we have not included the steps following the fitting of the linear model, which might include for example: 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 coded as missing values.

We can call this dataset directly 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 several `-9` to indicate missing data, but `NA` or other representations of missing are also valid within `ASRgenomics`.

A reasonable initial step is to proceed 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 for downstream analyses to facilitate further calculations. For this, the function `qc.filtering()` can be used as shown below:

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

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

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

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

## A total of 1651 markers were removed because their MAF was smaller than 0.05.

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

## Final cleaned marker matrix M contains 644 individuals and 3068 markers.
```

In this case, we proceed to remove markers according to some specifications. The options considered above were:

- `maf = 0.05` removes SNPs with minor allele frequence (MAF) below 0.05.
- `ind.callrate = 0.2` removes individuals with 20% or more missing values.
- `marker.callrate = 0.2` removes SNPs with 20% or more missing values.

This function reports that 11 individuals were dropped, and a total of 156 + 1,651 markers were eliminated. In addition, a new cleaned matrix, `M.clean`, was generated, and it can be explored with the commands:

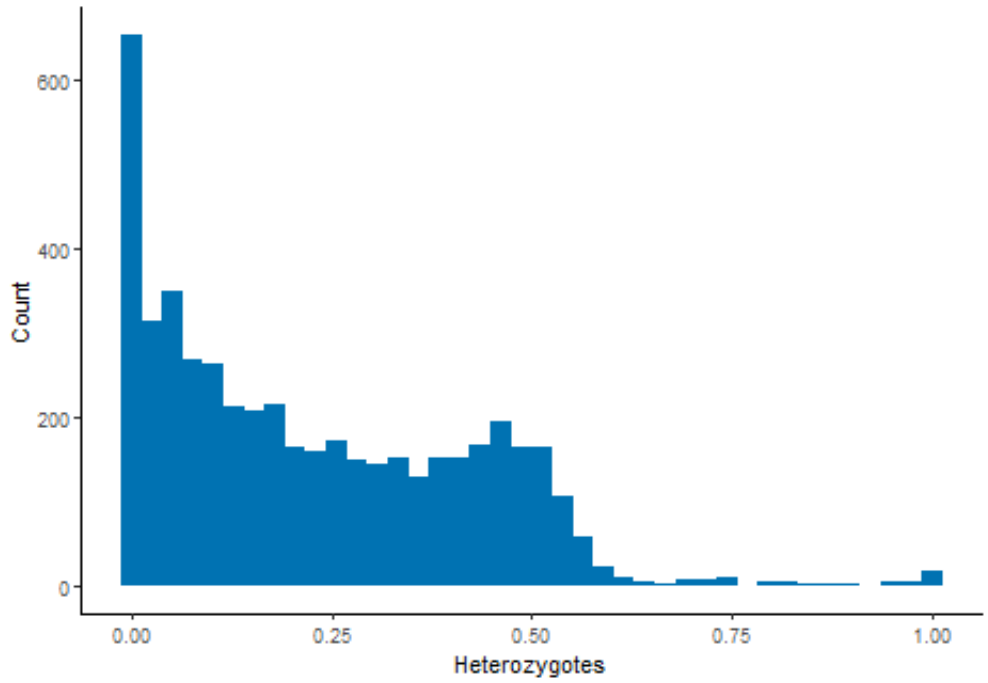
```
M_filter$M.clean[1:5, 1:5]
dim(M_filter$M.clean)
```

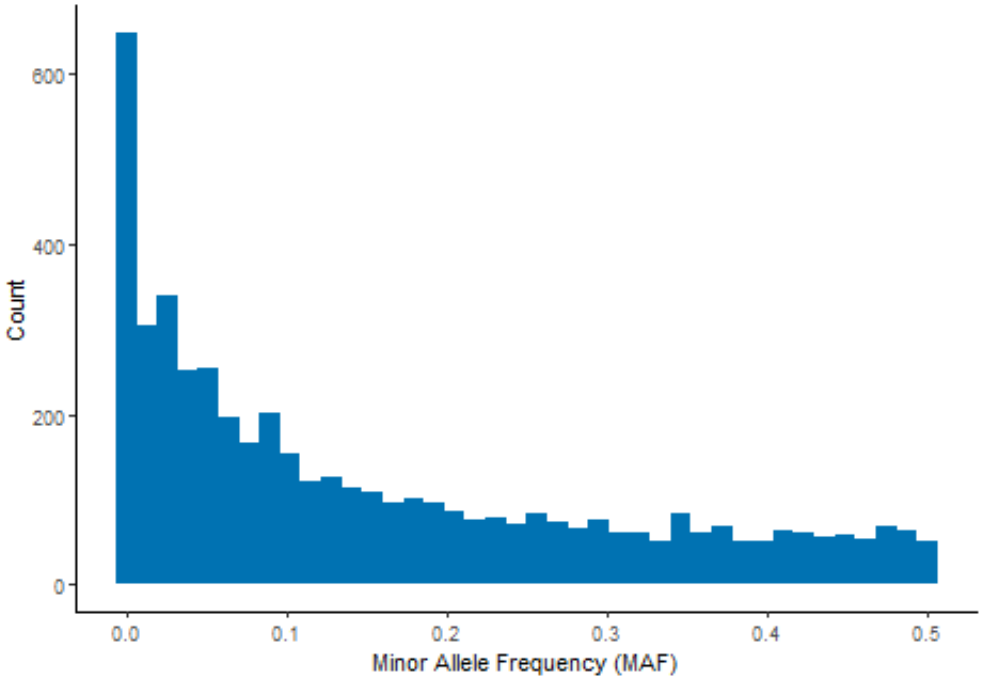
```
##         X0.10037.01.257 X0.10040.02.394 X0.10040.02.41 X0.10044.01.392 X0.10048.01.60
## 1085618               2               2              2               2              1
## 1091040               1               2              2               2              2
## 1091686               1   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

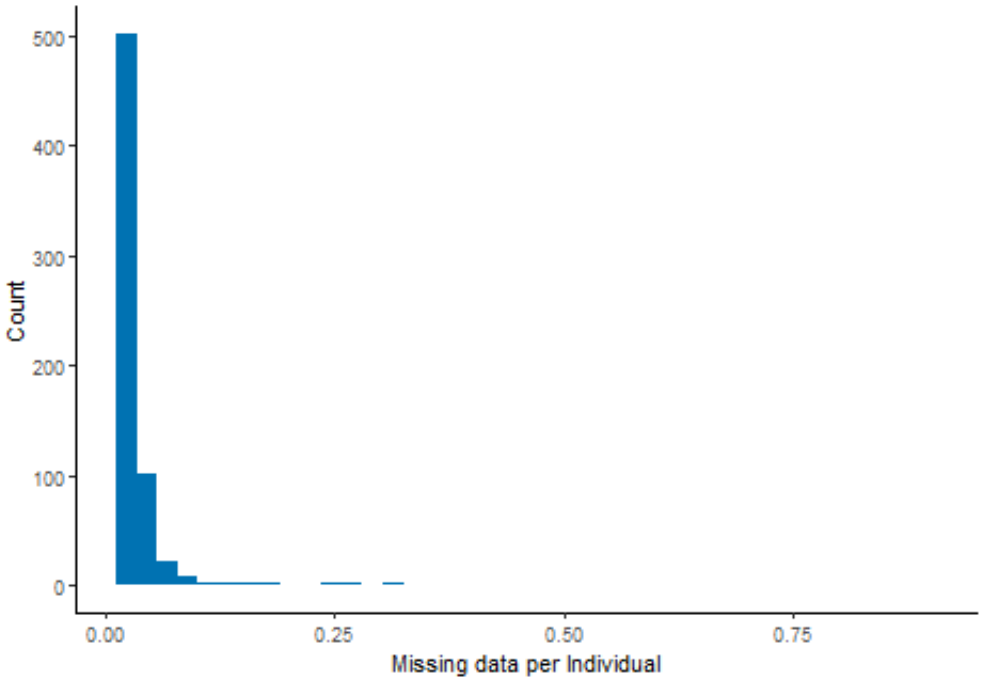
```
## [1]  644 3068
```

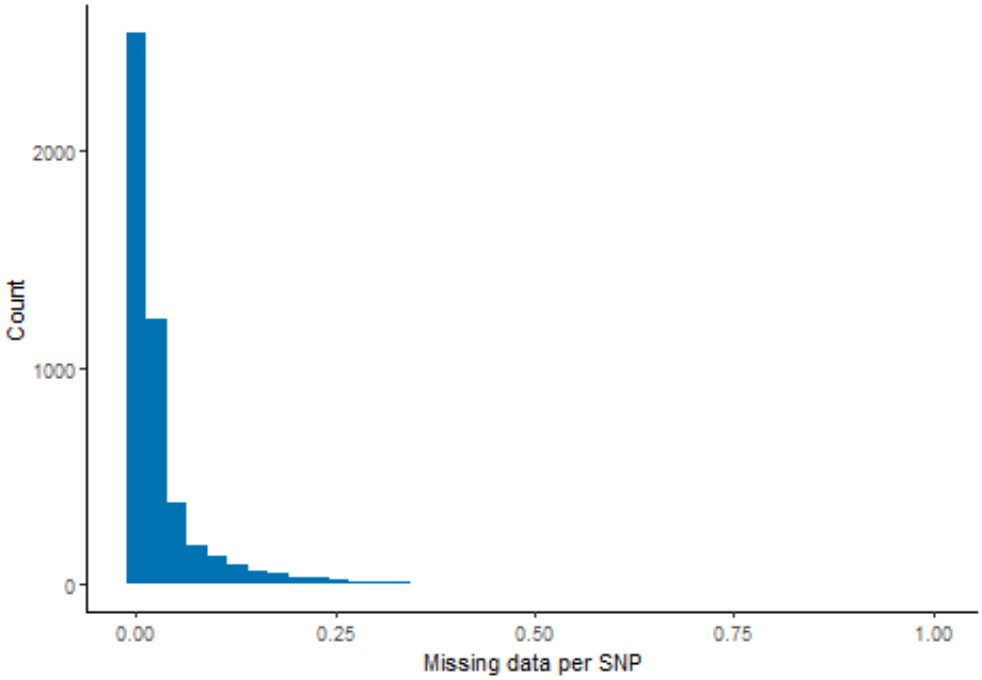
The dimension of this cleaned genomic matrix is 644 individuals by 3,068 markers. The same function produces four plots based on the original `geno.pine655` matrix, that can be further evaluated.

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




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


500
400
300
200
100
0
Count
0.00
0.25
0.50
0.75
Missing data per Individual



The `qc.filtering()` function also allows us to read and process matrices that are available in the bi-allele SNP data format (AA, AG, GG, CC, etc.). In addition, a simple imputation (average method) of missing values is implemented within the function, which is recommended whenever the proportion of missing values in the matrix is less than 5%. Further details can be found in the help.

4 Generating a Kinship Matrix

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

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

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

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

```
##           1085618   1091040    1091686   1082624    1088628
## 1085618  0.878450  0.038785  0.0369639 -0.028680  0.0862593
## 1091040  0.038785  0.855108  0.0500948 -0.038435  0.1390885
## 1091686  0.036964  0.050095  0.8477478 -0.032047 -0.0067241
## 1082624 -0.028680 -0.038435 -0.0320469  0.936679 -0.0306577
## 1088628  0.086259  0.139089 -0.0067241 -0.030658  0.9017011
```

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"`). 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 this later. This matrix requires the pedigree, which is loaded below and a portion is presented:

```
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
```

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

```
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 example 3,068 SNPs), and therefore it is prone to have sampling errors and other inconsistencies. These are minimized if only a small portion of the data is missing, and therefore they are reduced with adequate filtering, as we illustrated previously.

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

```
check_G <- kinship.diagnostics(K = G)
```

```
## Matrix dimension is: 644x644

## Rank of matrix is: 643

## Range diagonal values: 0.78853 to 1.01677

## Mean diagonal values: 0.91947

## Range off-diagonal values: -0.16312 to 0.61638

## Mean off-diagonal values: -0.00143

## There are 2 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
```

There are several diagnostic reports on this matrix that include range and mean values of the diagonal and off-diagonal values. Here, these ranges seem reasonable but two observations were reported as extreme on the diagonals, and no duplicate genotypes were reported. Thresholds for these flagged values can be modified within the function.

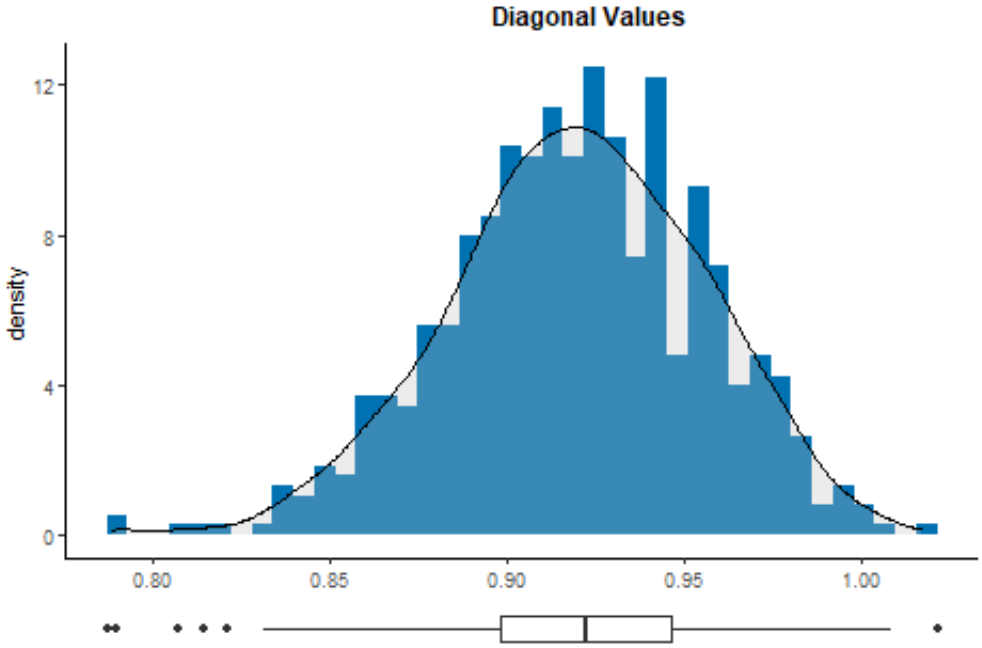
In our example, the flagged elements of the diagonal can be listed with:

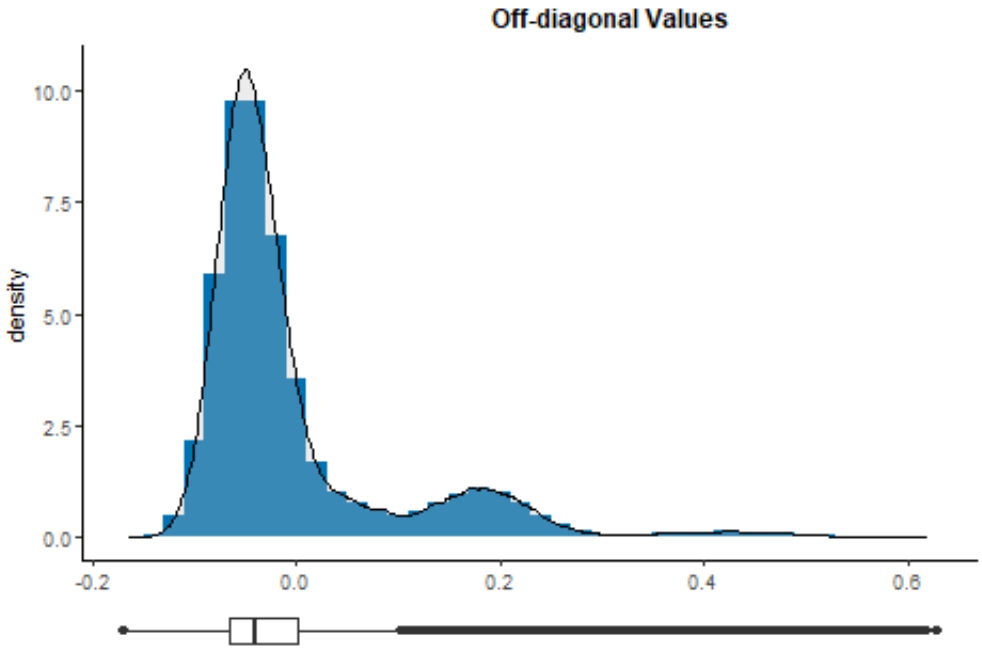
```
check_G$list.diagonal
```

```
##           value
## 1088926 0.79124
## 1091064 0.78853
```

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, but note that it is centred in ~ 0.92, that is smaller 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 they should be centred at 0, 0.25 and 0.50, respectively, but this is not happening with our genomic matrix **G**.

`ASRgenomics` has a few additional tools to help identify issues in this or other kinship matrices. For example, the function `matchG2A()` can be used to compare the values from the pedigree-

against the genomic-based relationship matrices, and this is something we will explore further in the next section.

6 Tuning a Genomic-based Kinship Matrix

Clearly, as we noted in the above two plots, our previous genomic matrix **G** has some clear bias, with genomic relationships underestimated from what is expected based on pedigree information. This bias is not rare in any genomic matrix given that 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 of the sample of individuals from our current population.

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

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

The `align` option from the `G.tuneup()` function, 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. Further details on this adjustment can be found in Christensen et al. (2012).

However, before proceeding, we need the portion of the individuals from the **A** matrix (that has a dimension of 2,034 × 2,034) that are present in our **G** matrix (with a dimension of 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 matrices we need:

```
dim(G2A$Aclean)
dim(G2A$Gclean)
```

```
## [1] 644 644

## [1] 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 genotyes. If we had genotypes missing on **G** but present on **A**, these will all be dropped, and vice versa.

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

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

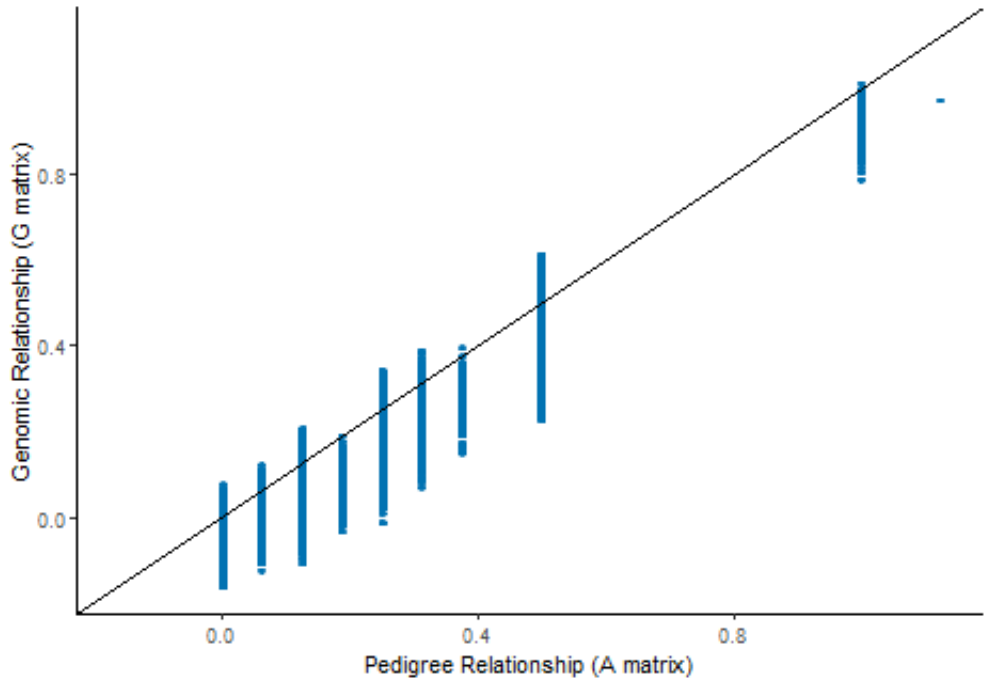
```
##         1087776 1087786 1087806 1087808 1087810
## 1087776     1.0     0.5     0.0     0.0     0.0
## 1087786     0.5     1.0     0.0     0.0     0.0
## 1087806     0.0     0.0     1.0     0.5     0.5
## 1087808     0.0     0.0     0.5     1.0     0.5
## 1087810     0.0     0.0     0.5     0.5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

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

```
##           1087776  1087786   1087806   1087808   1087810
## 1087776  0.900264  0.34872 -0.091128 -0.080647 -0.065307
## 1087786  0.348717  0.93348 -0.129594 -0.128653 -0.111678
## 1087806 -0.091128 -0.12959  0.869057  0.402765  0.438860
## 1087808 -0.080647 -0.12865  0.402765  0.871399  0.344164
## 1087810 -0.065307 -0.11168  0.438860  0.344164  0.909424
```

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

```
G2A$plotG2A
```



Note in this plot that values are concentrated on vertical lines for the **A** matrix, with reasonable ranges on the **G** matrix.

Also, this function produces an output matrix with these pairs, that if requested, can help to identify errors or inconsistencies between these matrices. The first few records in this example are called and shown below:

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

```
##   Row Col   IDRow   IDCol AValue  GValue  absdiff Diag
## 1   1   1 1080008 1080008    1.0 0.94356 0.056442    1
## 2   2   1 1080024 1080008    0.5 0.39224 0.107756    0
## 3   2   2 1080024 1080024    1.0 0.97826 0.021741    1
## 4   3   1 1080030 1080008    0.5 0.42687 0.073128    0
## 5   3   2 1080030 1080024    0.5 0.47762 0.022385    0
## 6   3   3 1080030 1080030    1.0 0.98658 0.013419    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 done with this dataset, for example we can explore the observations that have differences larger than 0.25, and these can be displayed with:

```
G2A$RM[G2A$RM$absdiff > 0.25, ]
```

```
##        Row Col   IDRow   IDCol AValue    GValue absdiff Diag
## 20095  200 195 1084642 1084618   0.50  0.242694 0.25731    0
## 21727  208 199 1084732 1084632   0.50  0.234499 0.26550    0
## 21936  209 200 1084734 1084642   0.50  0.242567 0.25743    0
## 58734  343  81 1087776 1081588   0.25 -0.012442 0.26244    0
## 78997  397 391 1088664 1088626   0.50  0.241472 0.25853    0
## 128774 507 503 1091028 1091014   0.50  0.227690 0.27231    0
## 155248 557 402 1093214 1088690   0.25 -0.011183 0.26118    0
```

From the above output 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` ~ 0.25). In addition, some individuals were declared as half-sibs (`AValue` $= 0.25$) when in fact they are unrelated (`GValue` ~ 0). Hence, these results indicate that some pedigree corrections might be required. We recommend a careful assessment, as these differences reflect errors on pedigree or DNA samples, and they are likely to affect downstream analyses.

Finally, using the clean `Gclean` and `Aclean` matrices from above, we are ready to perform our alignment, which 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: 1.25598131702587e-11

## Determinant for original matrix is: 3.6852912273057e-279

## Matrix was ALIGNED.

## Reciprocal conditional number for tune-up matrix is: 0.00018994846169216
```

Some of the reports on this function are in reference to the stability of this matrix. They are associated with the later inversion of the generated tuned-up matrix. For the reciprocal conditional number, values near zero are associated with an ill-conditioned matrix. For moderate size matrices, the determinant is also calculated, where again values at or near zero are associated with a singular or near-singular matrix, respectively. In our aligned matrix,

`G_align`, there is an improvement in the reciprocal conditional number. 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 matrix and its aligned version:

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

```
##           1087776  1087786   1087806   1087808   1087810
## 1087776  0.900264  0.34872 -0.091128 -0.080647 -0.065307
## 1087786  0.348717  0.93348 -0.129594 -0.128653 -0.111678
## 1087806 -0.091128 -0.12959  0.869057  0.402765  0.438860
## 1087808 -0.080647 -0.12865  0.402765  0.871399  0.344164
## 1087810 -0.065307 -0.11168  0.438860  0.344164  0.909424
```

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

```
##           1087776   1087786   1087806   1087808   1087810
## 1087776  0.980461  0.413727 -0.038229 -0.027459 -0.011696
## 1087786  0.413727  1.014591 -0.077754 -0.076787 -0.059345
## 1087806 -0.038229 -0.077754  0.948395  0.469263  0.506352
## 1087808 -0.027459 -0.076787  0.469263  0.950802  0.409049
## 1087810 -0.011696 -0.059345  0.506352  0.409049  0.989874
```

As expected, there are differences between these matrices, but the aligned matrix `G_align` is closer to our expected values. In this case, it is better to perform another set of diagnostics on this aligned matrix with:

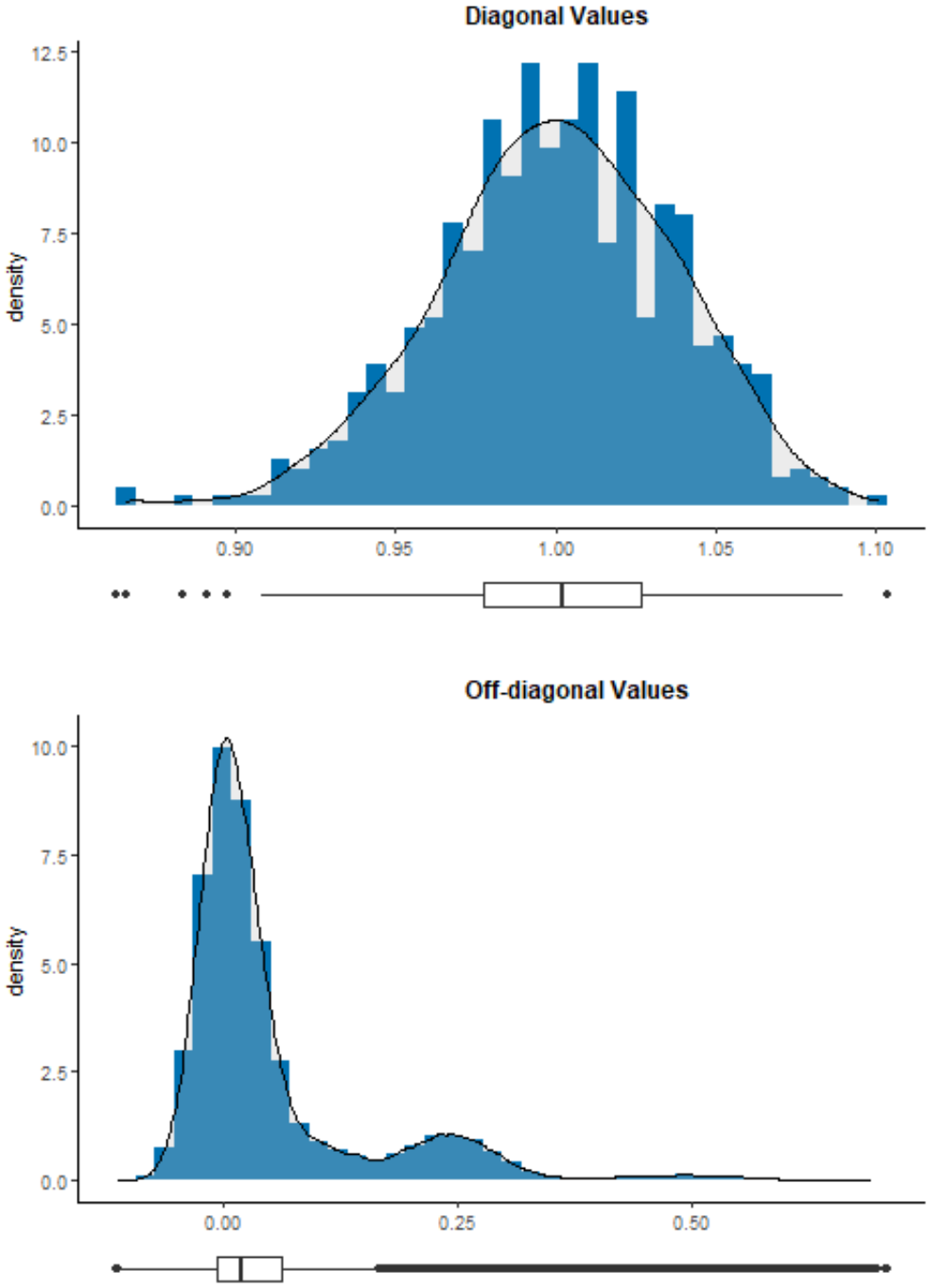
```
check_G_align <- kinship.diagnostics(K = G_align)
```

```
## Matrix dimension is: 644x644

## Rank of matrix is: 644

## Range diagonal values: 0.86565 to 1.10017

## Mean diagonal values: 1.00019

## Range off-diagonal values: -0.1122 to 0.68876

## 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
```

The first thing to note is that now there are no reported extreme values or possible duplicates. This is an indication that this matrix has improved properties, but further exploring and cleaning might still be required.

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 it has its three modes now centred at approximately 0, 0.25 and 0.5, as we expect given the pedigree information. Hence, we will use this matrix for downstream analyses.

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, then an identity matrix

of the same dimensions is used. 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.

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

7 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 to the phenotypic data in what is known as Genomic-BLUP (or G-BLUP). 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 model we require proper *communication* between both 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 linear mixed 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 (DEBV) for the trait diameter at breast height (DBH) from trees growing in site Nassau at age 6, corresponding to 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 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 `pheno.pine`. In our case, we need the matched individuals from the phenotypic data, that are listed under the output:

```
pheno.G$matchesP
```

and they are a total of 644 elements. Finally, 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()` if requested using the option `clean = TRUE`, also provides a $\mathbf{G}$ matrix with only the subset of individuals/genotypes that match the phenotypic data. This is useful where a reduced animal model is desired, as we will be able to fit our LMM only for those individuals with phenotypic data.

Given that we have our phenotypic dataset ready and our $\mathbf{G}$ matrix all prepared and checked (`G_align` in this case), we can now proceed to use `ASReml-R v.4` to fit our model. `ASReml-R` can accept both the $\mathbf{G}$ matrix or its inverse in full or sparse form. However, we strongly recommend you to supply the inverse of $\mathbf{G}$, *i.e.* $\mathbf{G}^{-1}$, and in sparse form. This is because obtaining the $\mathbf{G}^{-1}$ previously allowed us to assess its stability and to avoid ill-conditioning before the model is fitted. In addition, a sparse form matrix will require fewer computer resources as only the non-zero lower-diagonal of the relevant matrix is stored. To obtain this, or any other inverse, we use the function `G.inverse()`. For our example this looks like:

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

```
## Reciprocal conditional number for original matrix is: 0.00018994846169216
```

```
## Reciprocal conditional number for inverted matrix is: 0.000194169301255658
```

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

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

```
##      Row Col     Value
## [1,]   1   1  3.888191
## [2,]   2   1  0.155381
## [3,]   2   2  3.952043
## [4,]   3   1 -0.047951
## [5,]   3   2 -0.203150
## [6,]   3   3  4.624969
```

The report tells us that the reciprocal conditional number is small, but not too small to indicate problems, and this is also confirmed by the message that the matrix does not appear

to be ill-conditioned. In addition, we used the option `sparseform = TRUE` to obtain it in sparse form.

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

As with `G.tuneup()`, the function `G.inverse()` includes the tune up options of *blending* and *bending*.

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 the sparse form:

```
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
```

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

Using `ASReml-R v.4` we can now proceed to fit our LMM based on an animal model with the following code using the response variable `DBH_Adj`. But first, we will load the library and define factor terms using:

```
library(asreml)
```

```
## Warning: package 'asreml' was built under R version 4.0.3
```

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

and our LMM is:

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

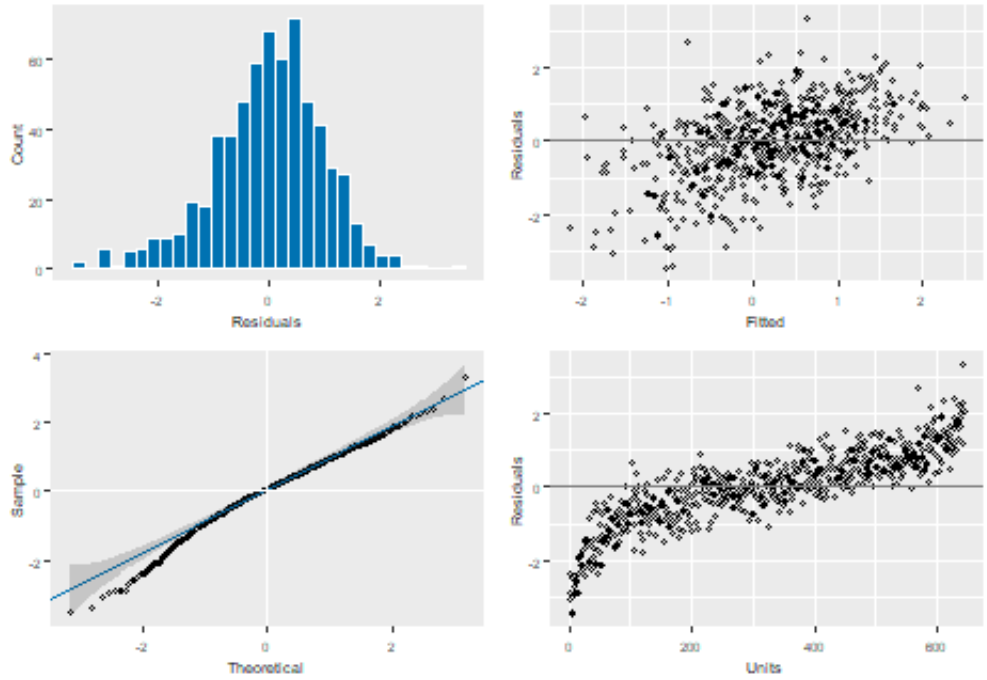
We are not going to explain this code in much detail, as this information is available directly from the `ASReml-R v.4` manual. However, in the above model we only have a single factor (`Genotype`) that is assumed to be random, and with the use of the command `vm()` we assign a variance model that links 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.

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

We can proceed to print out the estimated REML variance components and see the residual plots by using:

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

```
##                            component std.error z.ratio bound %ch
## vm(Genotype, Ginv.sparse)    1.1724   0.25019  4.6861     P   0
## units!units                  1.3832   0.13693 10.1015     P   0
## units!R                      1.0000        NA      NA     F   0
```



And we can calculate a genomic heritability for this analysis using:

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

```
##    Estimate       SE
## h2  0.45877 0.071454
```

This is a reasonable heritability for this trait with a small approximated standard error. Finally, some of the 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  0.7121417   0.67784  1.050611
## vm(Genotype, Ginv.sparse)_1080024 -0.4938816   0.67976 -0.726558
## vm(Genotype, Ginv.sparse)_1080030 -0.0071997   0.66536 -0.010821
## vm(Genotype, Ginv.sparse)_1080086  0.3037188   0.65765  0.461826
## vm(Genotype, Ginv.sparse)_1080116  0.0081954   0.67823  0.012084
## vm(Genotype, Ginv.sparse)_1080132  0.9046506   0.68444  1.321729
```

Issues with the molecular data and the preparation of the **G** matrix (and correspondingly its inverse) are likely to result in LMMs that do not converge. For this reason, we recommend careful checking of the genomic matrix, and use of the tools available for cleaning and tuning up, as we have done in this example.

It is also possible to use **G** matrices with slightly different tune up options (*e.g.*, blend with different proportions), and this is likely to yield different GP 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.

9 Generating/Exploring a Hybrid Genomic Matrix (H)

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

However, 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. In addition, we had a phenotypic dataset containing 861 individuals, of which 217 were dropped because of their lack of genomic data. This is clearly suboptimal, as 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 a pedigree dataset for a total of 2,032 individuals, which was also not considered in our GBLUP model, but 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 us to combine the information from the pedigree-based relationship matrix **A** together with the genomic-based **G** matrix to generate a so-called hybrid matrix **H**, and use 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 as shown before can be used directly with `ASReml-R v.4` 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 $\mathbf{G}^{-1}$. The function `H.inverse()` also 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 $\mathbf{G}^{-1}$. This is done by specifying the parameters λ, or the pair τ and ω. Further information on these scaling factors and their effects is available in Christensen and Lund (2009) and Legarra et al. (2009).

To illustrate this function we will continue using the pine dataset presented earlier, and here 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  3.888191  0.155381 -0.047951 -0.243349  0.24643
## 1080024  0.155381  3.952043 -0.203150  0.022449  0.18220
## 1080030 -0.047951 -0.203150  4.624969 -0.201870 -0.10016
## 1080086 -0.243349  0.022449 -0.201870  4.565396 -0.30496
## 1080116  0.246428  0.182202 -0.100158 -0.304959  4.13738
```

To get the $\mathbf{H}^{-1}$ matrix, 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 $\times$ 2,034 individuals. In this example, we have used a λ value of 0.9 for this scaling factor, 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 factors, 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.

Finally, the `H.inverse()` function can provide us with the **H** matrix directly. This might be computationally intensive, but it can be useful to explore the genetic relationship between some of the genotypes, particularly those not genotyped. This **H** matrix is obtained with the code:

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

The key aspect here is the use of the option `inverse = FALSE` that requests the **H** matrix directly. We can explore some of the differences between the **A** and **H** matrix 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 0.899722 0.059734 0.028601 0.067368     0 0.070957
## 44128 0.059734 0.949445 0.043637 0.033836     0 0.037493
## 50148 0.028601 0.043637 0.943465 0.065176     0 0.052514
## 50152 0.067368 0.033836 0.065176 0.970795     0 0.045642
## 51810 0.000000 0.000000 0.000000 0.000000     1 0.000000
## 52004 0.070957 0.037493 0.052514 0.045642     0 1.032929
```

```
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]
```

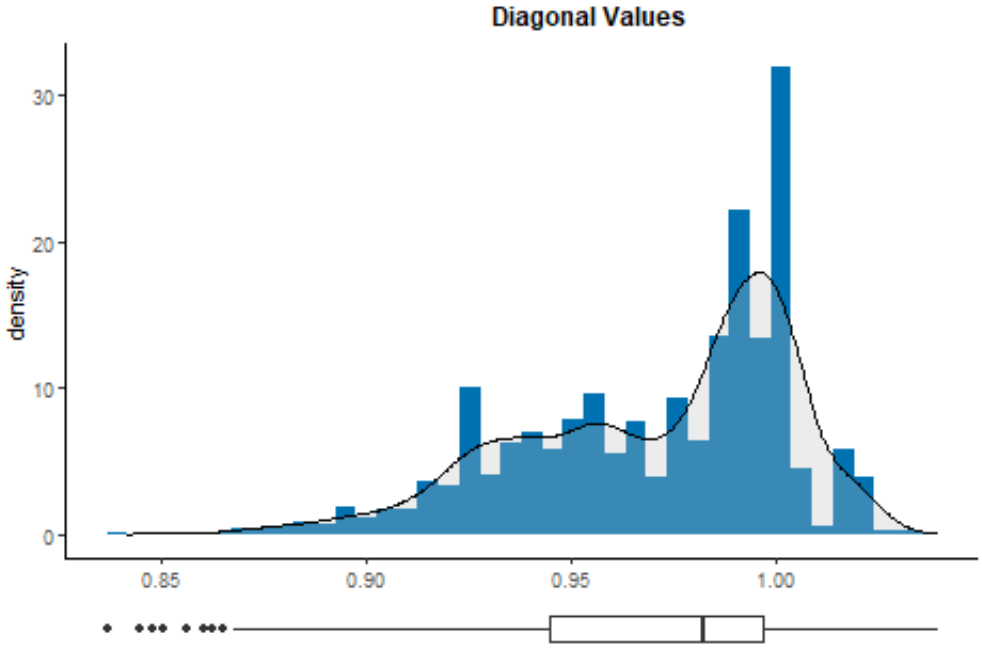
```
##            1087776   1087786   1087806   1087808    1087810
## 1087776  0.9140854  0.393489 -0.028267 -0.018212 -0.0061925
## 1087786  0.3934888  0.926320 -0.056754 -0.050002 -0.0410474
## 1087806 -0.0282666 -0.056754  0.906495  0.448462  0.4828010
## 1087808 -0.0182122 -0.050002  0.448462  0.899488  0.3943100
## 1087810 -0.0061925 -0.041047  0.482801  0.394310  0.9273327
```

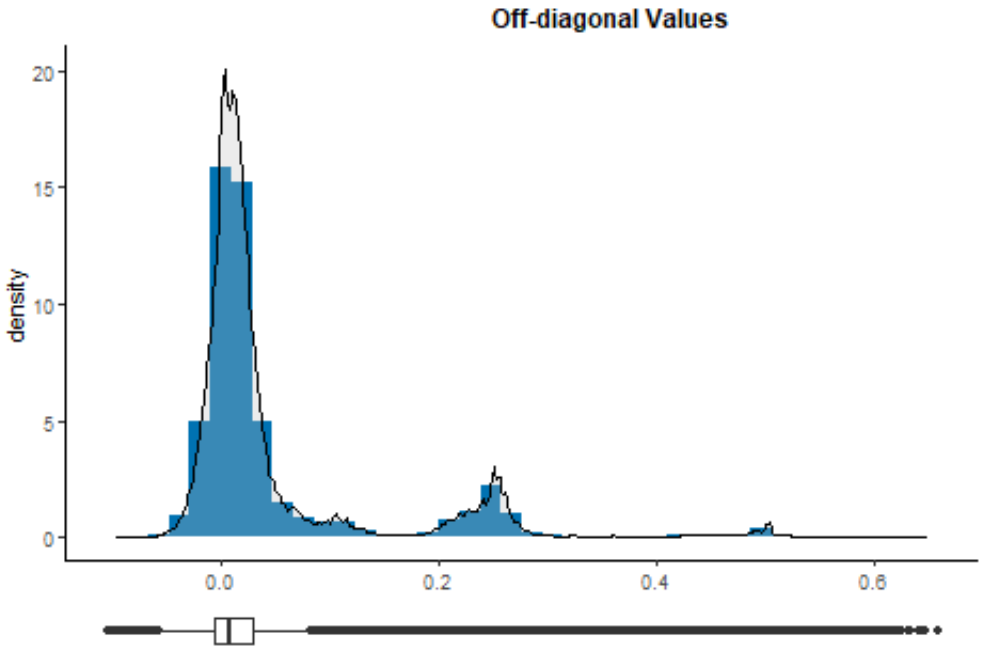
As with any other kinship matrices, we can proceed to assess the quality of either the **H** matrix or its inverse, by using our diagnostic tools within `ASRgenomics` as shown for **H** below:

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

```
## Matrix dimension is: 2034x2034

## Rank of matrix is: 2034

## Range diagonal values: 0.84167 to 1.0389

## Mean diagonal values: 0.97095

## Range off-diagonal values: -0.09586 to 0.64707

## Mean off-diagonal values: 0.0475

## 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
```

Interestingly, there are no extreme values reported here, and the plots obtained from this function using the **H** matrix are somehow 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.


Diagonal Values
density
30
20
10
0
0.85
0.90
0.95
1.00


Off-diagonal Values
density
20
15
10
5
0
0.0
0.2
0.4
0.6

10 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 v.4`. 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. It is generated with all the required attributes to be used directly in `ASReml-R`.

The following code is used to prepare our analysis:

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

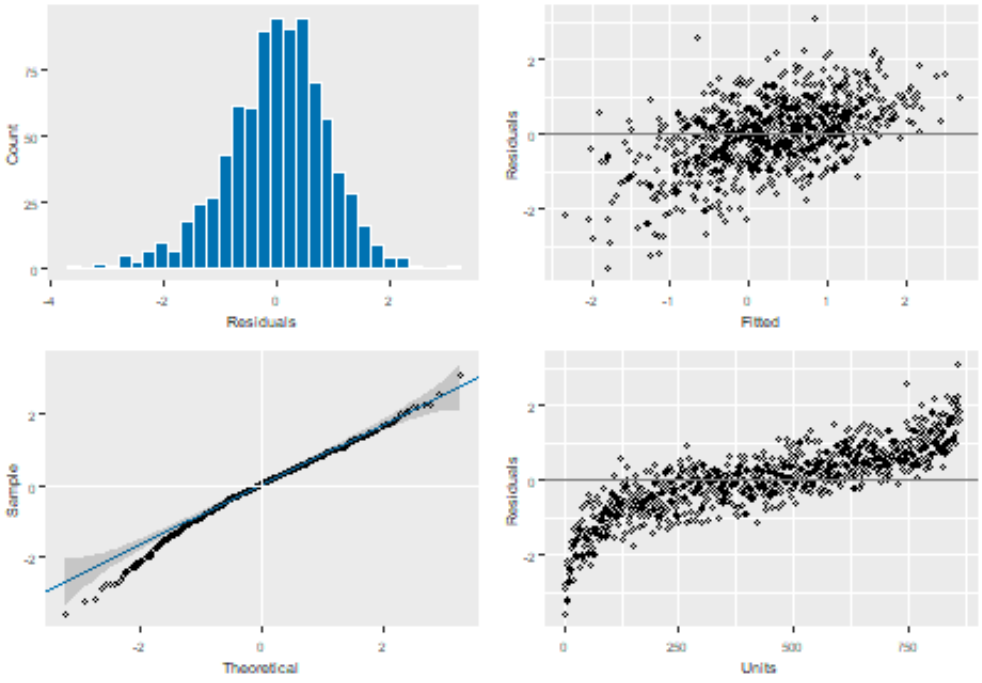
We can then proceed with the response variable `DBH_Adj` in this ssGBLUP analysis:

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

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

We can explore the residuals and estimated variance components with:

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



```
##                             component std.error z.ratio bound %ch
## vm(Genotype, Hinv.sparse)    1.3428   0.24901  5.3924     P   0
## units!units                  1.2439   0.12670  9.8176     P   0
## units!R                      1.0000        NA      NA     F   0
```

These components can be used to calculate a genomic heritability based on our **H** matrix as:

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

```
##    Estimate       SE
## h2  0.51912 0.067098
```

This is, as before, a good heritability for this trait with a larger value and slightly smaller approximated standard error.

Finally, we can extract GEBVs for the ssGLUP analysis that are shown below for a subset of genotypes together, for comparison, with the ones from our previous GBLUP model:

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

```
##                                  solution std.error   z.ratio
## vm(Genotype, Hinv.sparse)_1087776 0.5782045   0.66425 0.8704666
## vm(Genotype, Hinv.sparse)_1087786 0.7056700   0.65688 1.0742710
## vm(Genotype, Hinv.sparse)_1087806 0.0401140   0.67137 0.0597495
## vm(Genotype, Hinv.sparse)_1087808 0.0036799   0.66588 0.0055264
## vm(Genotype, Hinv.sparse)_1087810 0.2972410   0.66501 0.4469742
```

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

```
##                                  solution std.error   z.ratio
## vm(Genotype, Ginv.sparse)_1087776  0.406184   0.66738  0.608625
## vm(Genotype, Ginv.sparse)_1087786  0.509374   0.66083  0.770814
## vm(Genotype, Ginv.sparse)_1087806 -0.054528   0.67229 -0.081107
## vm(Genotype, Ginv.sparse)_1087808 -0.082718   0.66831 -0.123771
## vm(Genotype, Ginv.sparse)_1087810  0.083209   0.67064  0.124074
```

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

11 Additional Tools in ASRgenomics

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, such as Genome-wide Association Studies (GWAS).

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 was obtained from `ASRgenomics` using the function `G.matrix()` as shown before. The latter, can be generated from either the marker data or the genomic-based kinship matrix with the functions `snp.pca()` or `kinship.pca()`, respectively.

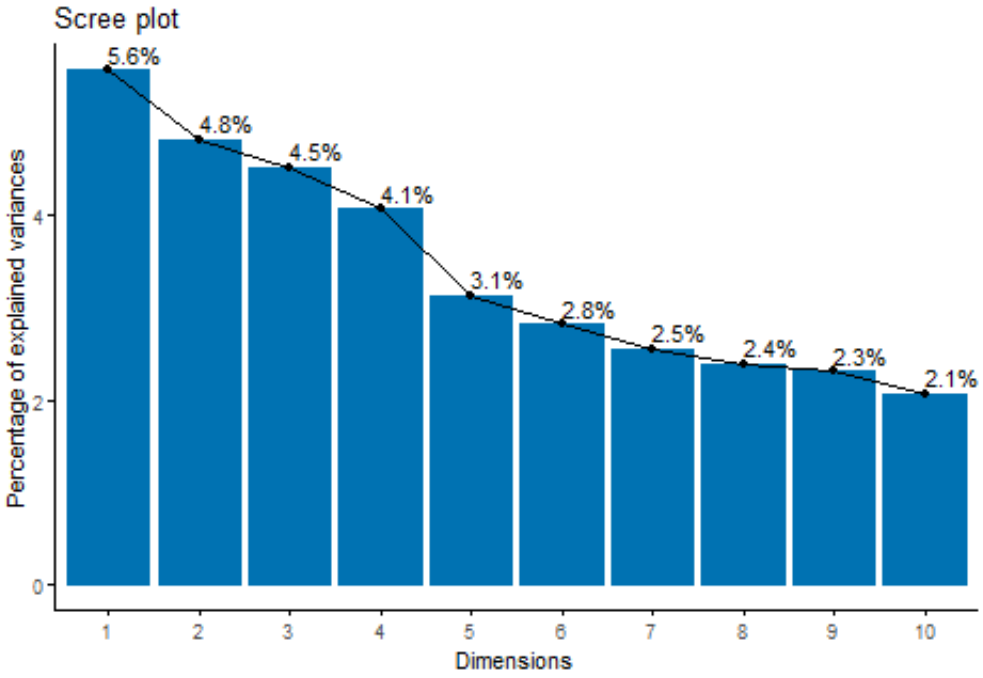
As an example, we illustrate this using the `kinship.pca()` function with the code presented below that makes use of our previously generated `G_align` matrix:

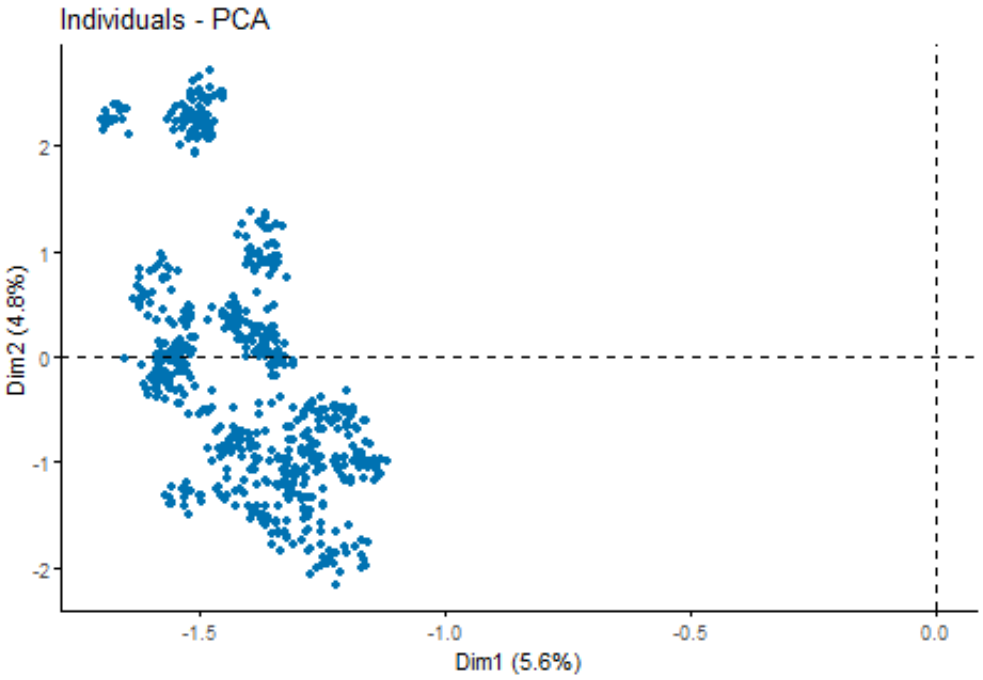
```
pca_pine <- kinship.pca(K = G_align, ncp = 10)
```

Here, we request only 10 eigenvectors (dimensions) with the option `ncp=10`, 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      35.781           5.5561                       5.5561
## Dim.2      30.988           4.8118                      10.3679
## Dim.3      29.043           4.5098                      14.8777
## Dim.4      26.124           4.0565                      18.9341
## Dim.5      20.097           3.1206                      22.0548
## Dim.6      18.138           2.8164                      24.8712
## Dim.7      16.409           2.5479                      27.4192
## Dim.8      15.386           2.3892                      29.8083
## Dim.9      14.835           2.3035                      32.1118
## Dim.10     13.335           2.0707                      34.1826
```





It is clear from the above plots that six dimensions explain almost 65% of the variability in the provided genomic matrix and that the grouping observed in the PCA plot is likely to be associated with the different sibships reflecting the structure of this population.

We requested 10 dimensions (or 10 PCs), and the scores of these PCs to be used in downstream GWAS analyses to form the **Q** matrix can be stored and accessed using:

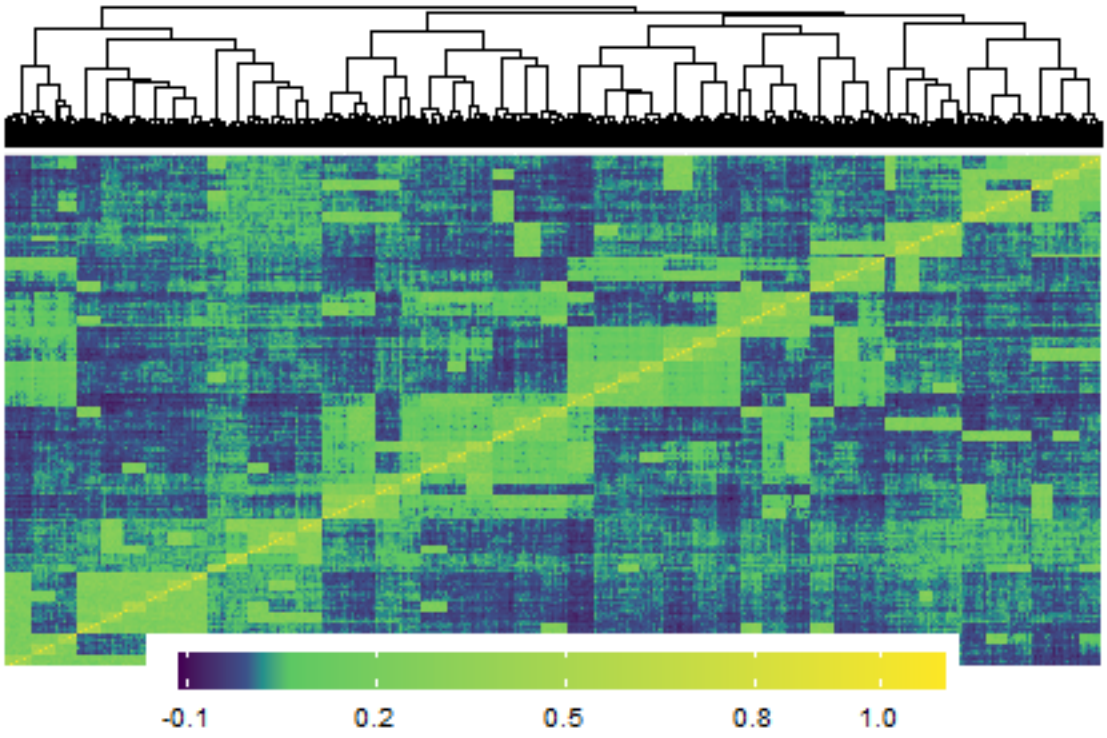
```
head(pca_pine$pca.scores)
```

```
##              PC1      PC2      PC3      PC4      PC5      PC6     PC7      PC8     PC9       PC10
## 1080008 -1.1716 -0.92041 -1.6446 -1.8925 -0.15672 -1.04157 0.46578 -0.39522 0.66045  0.046382
## 1080024 -1.1419 -1.03242 -1.6087 -2.0948 -0.28879 -1.03009 0.26418 -0.35953 0.55848 -0.022903
## 1080030 -1.1444 -1.02986 -1.5465 -2.0459 -0.33783 -1.02942 0.36192 -0.31783 0.72129 -0.126120
## 1080086 -1.2045 -1.04245 -1.5006 -1.9170 -0.27051 -1.05666 0.41103 -0.40032 0.59732 -0.058706
## 1080116 -1.1779 -0.90936 -1.6036 -1.9846 -0.28335 -0.78121 0.50575 -0.41716 0.66770 -0.018931
## 1080132 -1.1375 -1.03260 -1.5348 -1.9070 -0.13969 -0.91220 0.51544 -0.21823 0.74100  0.077045
```

The library `ASRgenomics` contains further utilitary functions, such as the `kinship.heatmap()` that is useful to display and explore genomic matrices. This function produces an enhanced heatmap plot based on a provided kinship matrix together with a dendogram. This plot is useful to identify graphically 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)
```



Other functions are available within `ASRgenomics`, including `G.predict()` to generate conditional predictions of random variables and others 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.

12 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 tutorial. A better understanding of the correct procedure to use and 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., Cellon, C., Olmstead, J. W., Garcia, A. A., Resende Jr, M. F., and Muñoz, P. R. 2016. AGHmatrix: R package to construct relationship matrices for autotetraploid and diploid species: A blueberry example. The Plant Genome. 9:1–10.

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

Christensen, O. F., and Lund, M. S. 2009. Genomic relationship matrix when some animals are not genotyped. In *Proc. 60th annual meeting EAAP. Wageningen press*, p. 299.

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

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

Martini, J. W., Schrauf, M. F., Garcia-Baccino, C. A., Pimentel, E. C., Munilla, S., Rogberg-Muñoz, A., 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. 50:1–9.

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

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

Resende, M., Muñoz, P., Resende, M. D., Garrick, D. J., Fernando, R. L., Davis, J. M., et al. 2012. Accuracy of genomic selection methods in a standard data set of loblolly pine (*Pinus taeda* l.). Genetics. 190:1503–1510.