The **GHap** Package (Version 1.2.1)

Yuri Tani Utsunomiya and Marco Milanesi

9 Jun 2016

Abstract

The **GHap** R package was designed to call haplotypes from phased SNP data. Given user-defined haplotype blocks (HapBlock), the package identifies the different haplotype alleles (HapAllele) present in the data and scores sample haplotype allele genotypes (HapGenotype) based on copy number (i.e., 0, 1 or 2 copies). **GHap** is an acronym for **G**enome-wide **Hap**lotyping, and is pronounced as **G-Hap**, not gap (although it is intended to fill the gap of haplotype analyses).

Importing phased data

An example input file can be created using the command:

```
# Copy the example data in the current working directory
library(GHap)
ghap.makefile()
```

These genotypes derive from the International HapMap Project Phase 3 (The International HapMap 3 Consortium, 2010), and comprise 1,011 subjects (from 11 populations) and 20,000 SNPs (randomly sampled from chromosome 2) mapped to the NCBI build 36 (hg18) assembly. The supported format is composed of three files with suffix:

.samples = space-delimited file without header containing two columns: Population and ID. Please notice that the Population column serves solely for the purpose of grouping samples, so the user can define any arbitrary family/cluster/subgroup and use as a "population" tag.

.markers = space-delimited file without header containing five columns: Chromosome, Marker, Position (in bp), Reference Allele (A0) and Alternative Allele (A1). Markers should be on a single chromosome and sorted by position.

.phase = space-delimited file without header containing the phased genotype matrix. The dimension of the matrix is expected to be m x $2n$, where m is the number of markers and n is the number of individuals. Alleles must be coded as 0 and 1. No missing values are allowed.

See below an example of five individuals from the ASW population with phased genotypes for five markers on chromosome 2:

```
=============================================================================
|  .samples file  |      .markers file       |      .phase file      |
=============================================================================
|  ASW NA19904    |  2 rs13383216 18228 A G  |  1 1 1 1 1 1 1 1 1 1  |
|  ASW NA20340    |  2 rs13386087 24503 G T  |  0 0 0 0 0 0 0 0 0 0  |
|  ASW NA20297    |  2 rs10179984 33092 A G  |  1 0 1 0 0 0 0 0 1 1  |
|  ASW NA20281    |  2 rs300761   60074 A G  |  0 1 0 0 1 1 0 1 0 1  |
|  ASW NA20348    |  2 rs6749571  72820 C G  |  0 0 0 0 0 0 0 1 0 0  |
=============================================================================
```

This format is conveniently obtained with very little manipulation from the output of widely used phasing software, such as SHAPEIT2 (O'Connell et al., 2014). For instance, to format your SHAPEIT2 files with UNIX standard commands use:

```
tail -n +3 shapeit2_file.sample | cut -d' ' -f1,2 > GHapfile.samples
cut -d' ' -f1-5 shapeit2_file.haps > GHapfile.markers
cut -d' ' -f1-5 --complement shapeit2_file.haps > GHapfile.phase
```

The `ghap.loadphase()` function is responsible for loading phased chromosomes from an input file and converting them into a native **GHap.phase** object. A detailed describtion of this object can be found in the documentation of the function. To load the example data in the package we can run:

```
#Load haplotype object
phase <- ghap.loadphase(
  samples.file = "human.samples",
  markers.file = "human.markers",
```

```
  phase.file = "human.phase"
)
#   Reading in marker map information... Done.
#   A total of 20000 markers were found for chromosome 2.
#   Reading in sample information... Done.
#   A total of 1011 individuals were found in 11 populations.
#   Reading in phased genotypes... (may take a few minutes for large datasets)
#   Your GHap.phase object was successfully loaded without apparent errors.
```

Subsetting, merging and exporting GHap.phase objects

The `ghap.subsetphase()` function can take any combination of markers and individuals and subset the **GHap.phase** object. This is achieved by setting undesired markers and individuals to **FALSE**. Inactivated individuals and markers are then ignored by all other functions taking a **GHap.phase** object as input.

For instance, we know that markers with low polymorphic information content may result in rare HapAlleles. If downstream analyses do not benefit from rare HapAlleles (e.g., haplotype association), it may be beneficial to prune these markers out prior to haplotyping. The code below shows how to subset 3,000 randomly selected markers with a minor allele frequency of at least 2%:

```
# Subset data - randomly select 3000 markers with maf > 0.02
maf <- ghap.maf(phase, ncores = 2)
set.seed(1988)
markers <- sample(phase$marker[maf > 0.02], 3000, replace = FALSE)
phase <- ghap.subsetphase(phase, unique(phase$id), markers)
#   Subsetting 1011 samples and 3000 markers... done.
#   Final data contains 1011 samples and 3000 markers.
```

It is also possible to merge two distinct **GHap.phase** objects. There are three possible merging tasks:

1 - Objects 1 and 2 have the same set of markers but different individuals

2 - Objects 1 and 2 have different sets of markers (with potential overlaps) but the same individuals

3 - Objects 1 and 2 have different sets of markers and individuals (with potential overlaps)

Currently, **GHap** only supports task 1. This is because phase information may not derive from a consensus marker panel in task 2, and task 3 has the additional problem of forcing missing genotypes. The example below shows how to bind two **GHap.phase** objects with different individuals:

```
# Select ASW and CEU individuals
ASW.ids <- unique(phase$id[phase$pop=="ASW"])
CEU.ids <- unique(phase$id[phase$pop=="CEU"])

# Subset data
phase.ASW <- ghap.subsetphase(phase = phase, ids = ASW.ids, markers = markers)
#    Subsetting 63 individuals and 3000 markers... Done.
#    Final data contains 63 individuals and 3000 markers.
phase.CEU <- ghap.subsetphase(phase = phase, ids = CEU.ids, markers = markers)
#    Subsetting 117 individuals and 3000 markers... Done.
#    Final data contains 117 individuals and 3000 markers.

# Merge phase.ASW and phase.CEU
phase.merge <- ghap.mergephase(phase.1 = phase.ASW, phase.2 = phase.CEU)
#    Creating the new GHap.phase object...
#    Your GHap.phase object was successfully merged without apparent errors.
```

GHap.phase objects can also be exported to text files:

```
# Output data
ghap.outphase(phase, "example")
#   Preparing example.markers... Done.
#   Preparing example.samples... Done.
#   Preparing example.phase... Done.
```

Defining HapBlocks

The user can provide the coordinates of any specific/arbitrary block to compute haplotype statistics. HapBlocks may also be defined based on LD structure. For instance, the algorithm created by Gabriel et al. (2002) implemented in PLINK v1.07/v1.9 (Purcell et al., 2007; Chang et al., 2015) can be used to estimate haplotype blocks based on the D' measure of LD. In **GHap**, we provide means to generate coordinates for HapBlocks based on sliding windows of markers. This strategy is particularly useful in genome-wide scans. We define HapBlocks here as user-defined sets of adjacent markers. An example of HapBlocks comprising ten markers, sliding five markers at a time, is provided below:

```
# Generate block coordinates based on windows of 10 markers, sliding 5 marker at a time
blocks <- ghap.blockgen(phase, 10, 5, "marker")
```

The user can also specify window and step size based on segment size in kilo-bases.

```
# Generate block coordinates based on windows of 500 kb, sliding 100 kb at a time
blocks <- ghap.blockgen(phase, 500, 100, "kb")
```

Haplotyping procedure

Let a haplotype library (HapLibrary) be the collection of observed HapAlleles for a given HapBlock. The haplotyping procedure implemented in **GHap** is straightforward: each HapAllele in the library is treated as a pseudo-marker, and HapGenotypes are scored as 0, 1 or 2 HapAllele copies. For instance, let's recall the example data provided in section 2:

```
=========================================================================
|  .samples file  |      .markers file       |      .phase file      |
=========================================================================
|  ASW NA19904    |  2 rs13383216 18228 A G  |  1 1 1 1 1 1 1 1 1 1  |
|  ASW NA20340    |  2 rs13386087 24503 G T  |  0 0 0 0 0 0 0 0 0 0  |
|  ASW NA20297    |  2 rs10179984 33092 A G  |  1 0 1 0 0 0 0 0 1 1  |
|  ASW NA20281    |  2 rs300761   60074 A G  |  0 1 0 0 1 1 0 1 0 1  |
|  ASW NA20348    |  2 rs6749571  72820 C G  |  0 0 0 0 0 0 0 1 0 0  |
=========================================================================
```

Let's assume the user wishes to call haplotypes for the HapBlock comprising the first three markers. The algorithm works as follows: First, we crop the matrix at the selected markers (for the sake of clarity, we will transpose the matrix and represent animals in rows and markers in columns):

```
POP ID      rs13383216 rs13386087 rs10179984
ASW NA19904 1          0          1
ASW NA19904 1          0          0
ASW NA20340 1          0          1
ASW NA20340 1          0          0
ASW NA20297 1          0          0
ASW NA20297 1          0          0
ASW NA20281 1          0          0
ASW NA20281 1          0          0
ASW NA20348 1          0          1
ASW NA20348 1          0          1
```

The HapLibrary is created based on the unique HapAlleles:

```
HapAllele1: 101 (GGG)
HapAllele2: 100 (GGA)
```

Then, for each HapAllele, individual HapGenotypes are scored based on the number of copies:

```
POP ID      GGG        GGA
ASW NA19904 1          1
ASW NA20340 1          1
ASW NA20297 0          2
ASW NA20281 0          2
ASW NA20348 2          0
```

The procedure is then repeated for each HapBlock. Below is an example with the test data, where HapGenotypes are called based on blocks of 10 markers with overlaps of 5 markers between blocks:

```
# Generate matrix of haplotype genotypes
ghap.haplotyping(phase, blocks, batchsize = 100, ncores = 2, freq = 0.05, outfile = "example")
#   Processing 599 blocks in:
#   1 batches of 99
#   5 batches of 100
#   599 blocks written to file
```

The user can control memory usage and process parallelization through the arguments *batchsize* and *ncores*, respectively. Notice that in the case above only HapAlleles with frequency > 0.05 will be retrieved. The function outputs three files with suffix:

.hapsamples = space-delimited file without header containing two columns: Population and Individual ID.

.hapalleles = space-delimited file without header containing five columns: Block Name, Chromosome, Start and End Position (in bp), and Haplotype Allele.

.hapgenotypes = space-delimited file without header containing the haplotype genotype matrix (coded as 0, 1 or 2 copies of the haplotype allele). The dimension of the matrix is m x n, where m is the number of haplotype alleles and n is the number of samples.

The example below was extracted from the first two HapBlocks for the HapMap data, using a random draw of 3,000 markers:

```
=============================================================================================
|  .hapsamples file  |            .hapalleles file            |      .hapgenotypes file      |
=============================================================================================
|     ASW NA19904    |  CHR2_B4 2 1009753 2462617 CCAATGTGGG  |           0 0 0 0 0          |
|     ASW NA20340    |  CHR2_B6 2 2511429 3071611 CCACACCAAT  |           0 0 0 0 0          |
|     ASW NA20297    |  CHR2_B6 2 2511429 3071611 CCACACCGAT  |           0 0 0 0 0          |
|     ASW NA20281    |  CHR2_B6 2 2511429 3071611 CTACACCAAT  |           0 0 1 0 0          |
|     ASW NA20348    |  CHR2_B6 2 2511429 3071611 CTACACCGAT  |           0 0 1 0 0          |
=============================================================================================
```

Importing HapAllele data

After HapAlleles have been scored, the data can be loaded into R using the `ghap.loadhaplo` function:

```
# Load haplotype genotypes
haplo <- ghap.loadhaplo("example.hapsamples", "example.hapalleles", "example.hapgenotypes")
#   Reading in haplotype allele information... Done.
#   A total of 1750 haplotype alleles were found.
#   Reading in sample information... Done.
#   A total of 1011 individuals were found in 11 populations.
#   Reading in haplotype genotypes... (may take a few minutes for large datasets)
#   Your GHap.haplo object was successfully loaded without apparent errors.
```

Subsetting, merging and exporting HapAllele data

Similar to the **GHap.phase** object, the user can also subset, merge and export **GHap.haplo** objects. For instance:

```
# Select CEU and ASW individuals
CEU.ids <- haplo$id[which(haplo$pop=="CEU")]
ASW.ids <- haplo$id[which(haplo$pop=="ASW")]

# Randomly select a set of HapAlleles
random.allele <- sample(x=c(TRUE, FALSE), size=haplo$nalleles, replace = TRUE)

# Subset data
haplo.CEU <- ghap.subsethaplo(haplo,ASW.ids,random.allele)
#    Subsetting 63 samples and 865 haplotype alleles... Done.
#    Final data contains 63 samples and 865 haplotype alleles.
haplo.ASW <- ghap.subsethaplo(haplo,CEU.ids,random.allele)
#    Subsetting 117 samples and 865 haplotype alleles... Done.
#    Final data contains 117 samples and 865 haplotype alleles.

# Merge haplo.CEU and haplo.ASW
haplo.merge <- ghap.mergehaplo(haplo.1 = haplo.CEU, haplo.2 = haplo.ASW, type = "individual")

# Output new GHap.haplo object
ghap.outhaplo(haplo = haplo.merge, outfile = "subset")
#    Preparing subset.hapsamples... Done.
#    Preparing subset.hapalleles... Done.
#    Preparing subset.hapgenotypes... Done.
```

HapAllele statistics

The statistics include absolute and relative frequencies, expected and observed number of homozygotes, and different tests for deficit of homozygotes in comparison to Hardy-Weinberg Equilibrium (HWE) expectations, which are described below.

Relative to HapAllele i, let p_i, h_i and n represent the relative frequency, the number of homozygotes, and the total number of observed chromosomes, respectively. Also, let S_i be some test statistic or score for the HapAllele, representing the goodness-of-fit of h_i to HWE expectations. The `ghap.hapstats()` function computes three candidate methods for S_i:

Method 1. The number of homozygotes for haplotype i is expected to be $E[h_i] = np_i^2$ under HWE. Provided we observed $O[h_i]$ homozygotes, deviations from HWE expectations can be expressed in terms of the expected-to-observed ratio:

$$S_i = \frac{E[h_i] + \alpha_1}{O[h_i] + \alpha_2}$$

where α_1 and α_2 are shrinkage parameters. The purpose of the shrinkage parameters is to regularize the scores towards a ratio of $\frac{\alpha_1}{\alpha_2}$, being particularly useful in cases where the number of observed homozygotes is close to zero. As the null ratio value is 1 (i.e., expected and observed counts are equal), a reasonable choice of shrinkage parameters is $\alpha_1 = \alpha_2 = 1$ (the default in **GHap**), which in practice introduces a bias equivalent to that of one additional expected and one additional observed homozygote. For a more detailed review on shrinkage expected-to-observed (or observed-to-expected) ratio, see Norén et al. (2013).

Method 2. Under the null hypothesis of HWE, $h_i \sim Binomial(n, p_i^2)$, with $E[h_i] = np_i^2$ and $VAR[h_i] = np_i^2(1 - p_i^2)$. Therefore, the probability of observing h_i or less homozygotes given the haplotype is in HWE is:

$$Pr(X \leq h_i) = \sum_{j \leq h_i} \binom{n}{j} p_i^{2j} (1 - p_i^2)^{n-j}$$

where X is a random draw from the Binomial distribution.

Method 3. Provided n is large, $h_i \sim Poisson(\lambda_i)$, where $\lambda_i = E[h_i] = VAR[h_i] = np_i^2$. This leads to probability:

$$Pr(X \leq h_i) = e^{-\lambda_i} \sum_{j \leq h_i} \frac{\lambda_i^j}{j!}$$

Note that the variance in the Binomial model is smaller than in the Poisson model, which in practice results in more conservative probabilities in the latter case. The summary statistics for the first two blocks in CEU can be obtained as:

```
CEU.hapstats <- ghap.hapstats(haplo, ncores = 2)
head(CEU.hapstats[,c(1,5:13)],n=5)
#       BLOCK     ALLELE  N       FREQ O.HOM O.HET     E.HOM     RATIO    BIN.logP    POI.logP
#   1 CHR2_B4 CCAATGTGGG 27 0.11538462     5    17 1.5576923 0.4262821 0.002160378 0.002319618
#   2 CHR2_B6 CCACACCAAT 20 0.08547009     1    18 0.8547009 0.9273504 0.102810150 0.102917993
#   3 CHR2_B6 CCACACCGAT  5 0.02136752     0     5 0.0534188 1.0534188 0.023204789 0.023199492
#   4 CHR2_B6 CTACACCAAT 11 0.04700855     0    11 0.2585470 1.2585470 0.112409787 0.112285539
#   5 CHR2_B6 CTACACCGAT  0 0.00000000     0     0 0.0000000 1.0000000 0.000000000 0.000000000
```

HapBlock statistics

The `ghap.blockstats` function summarizes HapAllele statistics per block and retrieves the expected heterozygosity and the number of alleles per HapBlock. For instance:

```
CEU.blockstats <- ghap.blockstats(CEU.hapstats, ncores = 2)
head(CEU.blockstats,n=2)
#       BLOCK CHR     BP1     BP2     EXP.H N.ALLELES
#   1 CHR2_B4   2 1009753 2462617 0.9866864         1
#   2 CHR2_B6   2 2511429 3071611 0.9900285         4
```

Please notice that when HapAlleles are prunned out by frequency the block statistics can retrieve high expected heterozygosity with small number of HapAlleles. The ideal situation would be computing both HapAllele and HapBlock statistics based on the entire set of HapAlleles.

HapAllele relationship matrix and PCA

Briefly, let $\mathbf{H}$ be the centered n x m matrix of HapGenotypes, where n is the number of observations and m is the number of HapAlleles. The HapAllele covariance among individuals can be computed as:

$$\mathbf{K} = q\mathbf{HDH}'$$

where $\mathbf{D} = \text{diag}(d_i)$, d_i is the weight of HapAllele i (default $d_i = 1$), and q is the inverse weighted sum of variances in the columns of $\mathbf{H}$. Notice that this is a generalization of the SNP-based genomic relationship matrix (VanRaden, 2008). The example below computes a kinship matrix from HapGenotypes and plots the first two eigenvectors of a singular value decomposition (principal components analysis) of this matrix.

```
# Reactivate all samples
haplo <- ghap.subsethaplo(haplo,haplo$id,haplo$allele.in)
#   Subsetting 1011 samples and 1750 haplotype alleles... done.
#   Final data contains 1011 samples and 1750 haplotype alleles.

# Compute Kinship matrix
K <- ghap.kinship(haplo, batchsize = 100)

# PCA analysis
pca <- ghap.pca(haplo,K)

# Plot
plot(x=pca$eigenvec$PC1, y=pca$eigenvec$PC2, xlab="PC1", ylab="PC2", pch="")
pop <- pca$eigenvec$POP
pop.col <- as.numeric(as.factor(pop))
pop <- sort(unique(pop))
legend("bottomright", legend = pop, col = 1:length(pop), pch = 1:length(pop), ncol = 3)
points(x=pca$eigenvec$PC1, y=pca$eigenvec$PC2,, pch = pop.col, col = pop.col, cex = 1.2)
```

Selection signatures analysis

Haplotype-based F_{ST} analyses are supported by the `ghap.fst()` function. Calculations are based on the multi-allelic formula

$$F_{ST} = (H_T - H_S)/H_T$$

where H_T is the total gene diversity (i.e., expected heterozygosity in the population) and H_S is the sub-population gene diversity (i.e., the average expected heterozygosity in the sub-populations). For instance, the example below compares the CEU and CHB populations for HapBlocks on chromosome 2:

```
# Compute haplotype allele statistics for each group
CHB.ids <- haplo$id[which(haplo$pop=="CHB")]
CEU.ids <- haplo$id[which(haplo$pop=="CEU")]
haplo <- ghap.subsethaplo(haplo,CHB.ids,haplo$allele.in)
#   Subsetting 84 samples and 1750 haplotype alleles... done.
#   Final data contains 84 samples and 1750 haplotype alleles.
CHB.hapstats <- ghap.hapstats(haplo,ncores = 2)
haplo <- ghap.subsethaplo(haplo,CEU.ids,haplo$allele.in)
#   Subsetting 117 samples and 1750 haplotype alleles... done.
#   Final data contains 117 samples and 1750 haplotype alleles.
CEU.hapstats <- ghap.hapstats(haplo,ncores = 2)
haplo <- ghap.subsethaplo(haplo,c(CHB.ids,CEU.ids),haplo$allele.in)
#   Subsetting 201 samples and 1750 haplotype alleles... done.
#   Final data contains 201 samples and 1750 haplotype alleles.
TOT.hapstats <- ghap.hapstats(haplo,ncores = 2)

# Compute haplotype block statistics for each group
CHB.blockstats <- ghap.blockstats(CHB.hapstats, ncores = 2)
CEU.blockstats <- ghap.blockstats(CEU.hapstats, ncores = 2)
TOT.blockstats <- ghap.blockstats(TOT.hapstats, ncores = 2)

# Calculate Fst
fst<-ghap.fst(CHB.blockstats, CEU.blockstats, TOT.blockstats)

# Plot results
top.fst <- fst[fst$FST == max(fst$FST, na.rm=TRUE),]
plot(
 x = (fst$BP1+fst$BP2)/2e+6,
 y = fst$FST, pch = "",
 ylab = expression(paste("Haplotype ", F[ST])),
 xlab = "Chromosome 2 (in Mb)",
 ylim=c(0,1)
)
abline(v=108.7, col="gray")
points(x = (fst$BP1+fst$BP2)/2e+6, y = fst$FST, pch = 20, col="#471FAA99")
points(x = (top.fst$BP1+top.fst$BP2)/2e+6, y = top.fst$FST, pch = 20, col="red")
text(x = 125, y = max(fst$FST, na.rm=TRUE), "EDAR", col="red")
CEU.hapstats[CEU.hapstats$BLOCK == top.fst$BLOCK & CEU.hapstats$FREQ > 0,1:9]
#           BLOCK CHR       BP1       BP2     ALLELE   N       FREQ O.HOM O.HET
#   893 CHR2_B291   2 108769202 108948789 TATTACCCCG 149 0.63675214    45    59
#   894 CHR2_B291   2 108769202 108948789 TATTACCCTA  14 0.05982906     0    14
CHB.hapstats[CHB.hapstats$BLOCK == top.fst$BLOCK & CHB.hapstats$FREQ > 0,1:9]
#           BLOCK CHR       BP1       BP2     ALLELE   N       FREQ O.HOM O.HET
#   892 CHR2_B291   2 108769202 108948789 TATCACCGTA   2 0.01190476     0     2
#   894 CHR2_B291   2 108769202 108948789 TATTACCCTA   2 0.01190476     0     2
#   895 CHR2_B291   2 108769202 108948789 TATTGCCCTA 155 0.92261905    71    13
```

Ideally, similar to the case of HapAllele and HapBlock statistics, the F_{ST} analysis should be carried out on the full set of HapAlleles, rather than a frequency-prunned subset.

Ancestry estimation

GHap offers a way to calculate the probability that a given HapAllele from a tested population was inherited from one of the tested parental populations. The procedure follows the method described by Bolormaa et al. (2011):

$$Pr(parent1) = \frac{p_{parent1}}{p_{parent1} + p_{parent2}} \text{ and } Pr(parent2) = \frac{p_{parent2}}{p_{parent1} + p_{parent2}}$$

where $p_{parent1}$ and $p_{parent2}$ are the HapAllele frequencies in the first and second parental populations, respectively. Assignments are performed as follows: if the probability of one of the parents exceeds a user-defined threshold (default = 0.60), the HapAllele origin is assigned to that parental population. Parental probabilities are set to zero if the HapAllele frequency in the parental populations are lower than a certain threshold (default = 0.05). For instance, using CEU and YRI as proxy parental populations for ASW, we could assign HapAlleles in ASW to CEU or YRI using the following code:

```
# Subset data - randomly select 3000 markers with maf > 0.02
ASW.ids <- unique(phase$id[phase$pop=="ASW"])
YRI.ids <- unique(phase$id[phase$pop=="YRI"])
CEU.ids <- unique(phase$id[phase$pop=="CEU"])
phase <- ghap.subsetphase(phase, c(ASW.ids,YRI.ids,CEU.ids), phase$marker)
#   Subsetting 295 samples and 20000 markers... done.
#   Final data contains 295 samples and 20000 markers.
maf <- ghap.maf(phase, ncores = 2)
set.seed(1988)
markers <- sample(phase$marker[maf > 0.02], 3000, replace = FALSE)
phase <- ghap.subsetphase(phase, c(ASW.ids,YRI.ids,CEU.ids), markers)
#   Subsetting 295 samples and 3000 markers... done.
#   Final data contains 295 samples and 3000 markers.

# Generate block coordinates based on windows of 10 markers, sliding 5 marker at a time
blocks <- ghap.blockgen(phase, 10, 5, "marker")

# Generate matrix of haplotype genotypes
ghap.haplotyping(phase, blocks, batchsize = 100, ncores = 2, freq = 0, outfile = "example")
#   Processing 599 blocks in:
#   1 batches of 99
#   5 batches of 100
#   599 blocks written to file

# Load haplotype genotypes
haplo <- ghap.loadhaplo("example.hapsamples", "example.hapalleles", "example.hapgenotypes")
#   Reading in haplotype allele information... Done.
#   A total of 65307 haplotype alleles were found.
#   Reading in sample information... Done.
#   A total of 295 individuals were found in 3 populations.
#   Reading in haplotype genotypes... Done.
#   Your GHap.haplo object was successfully loaded without apparent errors.

# Compute haplotype allele statistics for each group
```

```
haplo <- ghap.subsethaplo(haplo,YRI.ids,haplo$allele.in)
#   Subsetting 115 samples and 65307 haplotype alleles... done.
#   Final data contains 115 samples and 65307 haplotype alleles.
YRI.hapstats <- ghap.hapstats(haplo,ncores = 2)
haplo <- ghap.subsethaplo(haplo,CEU.ids,haplo$allele.in)
#   Subsetting 117 samples and 65307 haplotype alleles... done.
#   Final data contains 117 samples and 65307 haplotype alleles.
CEU.hapstats <- ghap.hapstats(haplo,ncores = 2)
haplo <- ghap.subsethaplo(haplo,ASW.ids,haplo$allele.in)
#   Subsetting 63 samples and 65307 haplotype alleles... done.
#   Final data contains 63 samples and 65307 haplotype alleles.
ASW.hapstats <- ghap.hapstats(haplo,ncores = 2)

# Find haplotype origin
# ASW is the test population. YRI and CEU are used as parental populations
# The frequency threshold is set to 0.05 and the probability of assignment to 0.60
ancestry <- ghap.ancestral(ASW.hapstats, YRI.hapstats, CEU.hapstats, 0.05, 0.60)
head(ancestry[ancestry$ORIGIN != "UNK" & ancestry$FREQ.TEST > 0,-c(2:4)])
#     BLOCK     ALLELE   FREQ.TEST FREQ.PARENT1 FREQ.PARENT2 PROB.PARENT1 PROB.PARENT2  ORIGIN
#   CHR2_B1 TCCCTGCTCC 0.047619048  0.052173913   0.00000000            1    0.0000000 PARENT1
#   CHR2_B1 TCTGTGCTCC 0.031746032  0.056521739   0.00000000            1    0.0000000 PARENT1
#   CHR2_B1 TTCCTACCCG 0.007936508  0.000000000   0.10683761            0    1.0000000 PARENT2
#   CHR2_B1 TTCCTACTCC 0.023809524  0.013043478   0.06837607            0    0.8397992 PARENT2
#   CHR2_B1 TTCCTGCCCC 0.031746032  0.004347826   0.05128205            0    0.9218437 PARENT2
#   CHR2_B1 TTCCTGTTCG 0.007936508  0.000000000   0.05555556            0    1.0000000 PARENT2
```

Linear mixed model analysis

The linear mixed model procedure in **GHap** is implemented under two different frameworks: 1 - Bayesian estimation via Gibbs sampling, inspired by similar models and algorithms described elsewhere (Wang et al., 1994; Sorensen et al., 1995; Pérez and de Los Campos, 2015); and 2 - frequentist estimation via maximum likelihood. Greater details of these frameworks can be found in the **Appendix**. The targeted model is:

$$\mathbf{y} = \mathbf{Xb} + \mathbf{Zu} + \mathbf{Zp} + \mathbf{e}$$

where $\mathbf{X}$ is a n x p matrix relating the n records in $\mathbf{y}$ to the p fixed effects in $\mathbf{b}$, $\mathbf{Z}$ is a n x m incidence matrix relating the n records in $\mathbf{y}$ to the m random effects in $\mathbf{u}$ (with variance-covariance $\mathbf{K}\sigma^2_u$) and $\mathbf{p}$ (with variance-covariance $\mathbf{I}\sigma^2_p$), and $\mathbf{e}$ is the vector of residuals (with variance-covariance $\mathbf{W}\sigma^2_e$). Here we assume residuals are independent, such that $\mathbf{W} = \text{diag}(w_i)$. Also, if we let $\mathbf{K}$ be the HapAllele relationship matrix, then $\mathbf{u}$ becomes the HapAllele-based polygenic effects/breeding values, and σ^2_u becomes the variance due to HapAlleles. Importantly, any arbitrary $\mathbf{K}$ matrix is admitted, such that one may fit models combining pedigree and haplotype relationships (e.g., single-step GWAS analysis). Although including permanent environmental effects $\mathbf{p}$ may be beneficial for the analysis of repeated measurements, the user can decide whether these should be included (the default model does not include these effects).

When variance components are known, solutions of fixed and random effects can be obtained with the `ghap.mme()` function using Henderson's mixed model equations:

$$\begin{bmatrix} \mathbf{X'W^{-1}X} & \mathbf{X'W^{-1}Z} & \mathbf{X'W^{-1}Z} \\ \mathbf{Z'W^{-1}X} & \mathbf{Z'W^{-1}Z} + \mathbf{K^{-1}}\frac{\sigma^2_e}{\sigma^2_u} & \mathbf{Z'W^{-1}Z} \\ \mathbf{Z'W^{-1}X} & \mathbf{Z'W^{-1}Z} & \mathbf{Z'W^{-1}Z} + \mathbf{I}\frac{\sigma^2_e}{\sigma^2_p} \end{bmatrix} \begin{bmatrix} \hat{\mathbf{b}} \\ \hat{\mathbf{u}} \\ \hat{\mathbf{p}} \end{bmatrix} = \begin{bmatrix} \mathbf{X'W^{-1}y} \\ \mathbf{Z'W^{-1}y} \\ \mathbf{Z'W^{-1}y} \end{bmatrix}$$

Otherwise, the Bayesian framework implemented in `ghap.blmm()` or the frequentist framework implemented in `ghap.lmm()` can be used to obtain estimates of model parameters. For the frequentist approach, the likelihood of the data given the parameters is assumed:

$$\mathbf{y} \mid \mathbf{b}, \mathbf{u}, \mathbf{p}, \sigma_u^2, \sigma_p^2, \sigma_e^2 \sim N(\mathbf{Xb}, \mathbf{V})$$

where $\mathbf{V} = \mathbf{ZKZ}'\sigma_u^2 + \mathbf{ZZ}'\sigma_p^2 + \mathbf{W}\sigma_e^2$. The negative log-likelihood is minimized using a quasi-Newton method with box constraints.

In the case of the Bayesian method, the likelihood of the data given the parameters is assumed:

$$\mathbf{y} \mid \mathbf{b}, \mathbf{u}, \mathbf{p}, \sigma_u^2, \sigma_p^2, \sigma_e^2 \sim N(\mathbf{Xb} + \mathbf{Zu} + \mathbf{Zp}, \mathbf{W}\sigma_e^2)$$

In this setting, the parameters we wish to estimate are $\mathbf{b}$, $\mathbf{u}$, $\mathbf{p}$, σ_u^2, σ_p^2 and σ_e^2, for which we assign the following prior distributions:

$$\begin{gathered}
\mathbf{b} \propto \text{constant} \\
\mathbf{u} \mid \sigma_u^2 \sim N(0, \mathbf{K}\sigma_u^2) \\
\mathbf{p} \mid \sigma_p^2 \sim N(0, \mathbf{I}\sigma_p^2) \\
\sigma_u^2 \sim \chi^{-2}(\nu_u, S_u^2) \\
\sigma_p^2 \sim \chi^{-2}(\nu_p, S_p^2) \\
\sigma_e^2 \sim \chi^{-2}(\nu_e, S_e^2)
\end{gathered}$$

The hyper-parameters ν_u, ν_p, ν_e, S_u^2, S_p^2 and S_e^2 are the random effects and residual variance degrees of freedom and scale parameters, respectively. Following Bayes' Theorem, we have:

$$\text{posterior} \propto \text{likelihood} \times \text{prior}$$

such that the full posterior distribution can be written as:

$$Pr(\mathbf{b}, \mathbf{u}, \mathbf{p}, \sigma_u^2, \sigma_p^2, \sigma_e^2 \mid \mathbf{y}) \propto Pr(\mathbf{y} \mid \mathbf{b}, \mathbf{u}, \mathbf{p}, \sigma_u^2, \sigma_p^2, \sigma_e^2) Pr(\mathbf{u} \mid \sigma_u^2) Pr(\mathbf{p} \mid \sigma_p^2) Pr(\sigma_u^2) Pr(\sigma_p^2) Pr(\sigma_e^2)$$

The `ghap.blmm()` function uses a Gibbs sampling algorithm to obtain samples from the marginal conditional posterior distribution of each parameter. The Markov Chain is ran for a user-specified number of Monte Carlo simulations, and estimates for the posterior means of the parameters are obtained by averaging over simulations. As the Makov Chain is allowed to walk over the entire domain of the posterior distribution, effects of local maxima are mitigated by averaging over simulations, such that parameter estimates may better represent the values under the global maxima.

Different from other GWAS tools, **GHap** accommodates repeated measurements through $\mathbf{Z}$ and heteroskedastic residuals through $\mathbf{W}$. The package can also deal with ordered categorical data as response under the Bayesian framework. This is achieved by assuming that categories emerge from thresholds of a latent normal variable (i.e., "liability"). We use data augmentation to sample thresholds and observations from the underlying latent variable, which we then treat as responses in the main algorithm.

In the example below we simulate a quantitative trait with 50% heritability, where a major HapAllele accounts for 30% of the genetic variance. We use the same data to illustrate a binary trait that emerged from thresholding an underlying quantitative trait.

```
# Quantitative trait with 50\% heritability
# One major haplotype accounting for 30\% of the genetic variance
sim <- ghap.simpheno(haplo = haplo, K = K, h2 = 0.5, g2 = 0.3, major = 1000,seed=1988)

# Binary trait from the previous example
# 0 if observation is below the 70\% percentile
# 1 otherwise
thr <- quantile(x = sim$data$phenotype, probs = 0.7)
sim$data$phenotype2 <- sim$data$phenotype
sim$data$phenotype2[sim$data$phenotype < thr] <- 0
sim$data$phenotype2[sim$data$phenotype >= thr] <- 1
```

In the example below, we fit the simulated phenotypes including only an intercept as fixed effect. If we assume known variances:

```
#Henderson's mixed model equations
model <- ghap.mme(fixed = phenotype ~ 1, random = "individual", data = sim$data, K = K
                  varcomp = c(sim$vare,sim$varu))
plot(model$u,sim$u, ylab="True Breeding Value", xlab="Estimated Breeding Value")
cor(model$u,sim$u)
```

With unkwown variances we have to estimate model parameters with either `ghap.lmm()` or `ghap.blmm()`. With `ghap.lmm()`, the model is fitted by running:

```
#Continuous model
model <- ghap.lmm(fixed = phenotype ~ 1, random = "individual", data = sim$data, K = K)
model$h2
plot(model$u,sim$u, ylab="True Breeding Value", xlab="Estimated Breeding Value")
cor(model$u,sim$u)
```

Please notice that currently `ghap.lmm()` can only handle quantitative traits. Alternatively, we can use `ghap.blmm()`. Here a single Markov chain is ran with 500 Monte Carlo simulations and a thinning interval of 1. Analyses of empirical data may require larger numbers, and we recommend using at least 100 simulations as burn-in, 1000 main simulations and a minimum thinning interval of 2. Also, running two or more parallel Markov chains may be beneficial to speed up computations and assess variance across independent runs (i.e., evaluate posterior convergence).

```
#Continuous model
model <- ghap.blmm(fixed = phenotype ~ 1, random = "individual", data = sim$data, K = K)
model$h2
plot(model$u,sim$u, ylab="True Breeding Value", xlab="Estimated Breeding Value")
cor(model$u,sim$u)

#Threshold model
model2 <- ghap.blmm(fixed = phenotype2 ~ 1, random = "individual",
                    ordinal = TRUE, data = sim$data, K = K)
model2$h2
plot(model2$u,sim$u, ylab="True Breeding Value", xlab="Estimated Breeding Value")
cor(model2$u,sim$u)
model2$thresholds[2]
quantile(x = scale(sim$data$phenotype), probs = 0.7)
```

Future implementations of **GHap** will consider more flexible models, especially regarding the definition of random effects.

GWAS treating HapAlleles as fixed effects

Consider the residuals from the linear mixed model analysis:

$$\hat{\mathbf{e}} = \mathbf{y} - \mathbf{X}\hat{\mathbf{b}} - \mathbf{Z}\hat{\mathbf{u}} - \mathbf{Z}\hat{\mathbf{p}}$$

These residuals can be viewed as adjusted records accounting for covariates, polygenic effects and permanent environmental effects, i.e. $\mathbf{y}_{\text{adj}} = \hat{\mathbf{e}}$. We can then conduct least squares regression to test each HapAllele at a time for association with phenotypes. The fixed effect, error variance and test statistic of a given HapAllele are estimated using least square equations:

$$\hat{a}_i = (\mathbf{h}_i'\mathbf{h}_i)^{-1}\mathbf{h}_i'\mathbf{y}_{\text{adj}}$$
$$VAR(\hat{a}_i) = (\mathbf{h}_i'\mathbf{h}_i)^{-1}\hat{\sigma}_e^2$$
$$t_i^2 = \frac{\hat{a}_i^2}{VAR(\hat{a}_i)}$$

Under the null hypothesis that the regression coefficient is zero $t_i^2 \sim \chi^2(\nu = 1)$. Given a **Ghap.blmm** object, the `ghap.assoc()` function regresses adjusted records on each HapAllele. As the linear mixed model admits repeated measures through $\mathbf{Z}$, the adjusted records are dully mapped against the vector of HapGenotypes in `ghap.assoc()`. The example below takes the simulated phenotype data and regress adjusted records agains HapAlleles.

```
#HapAllele GWAS
gwas.allele <- ghap.assoc(blmm = model, haplo = haplo, type = "HapAllele", gc = TRUE, ncores=2)

#Plot
gwas.allele$POS <- (gwas.allele$BP1+gwas.allele$BP2)/2e+6
plot(gwas.allele$POS,gwas.allele$logP, xlab="Chromosome 2 position (in Mb)",
     ylab=expression(-log[10](p)), pch=20, col="#471FAA99")
abline(v=(haplo$bp1[1000]+haplo$bp2[1000])/2e+6,lty=3)
```

The user must be aware of two known caveats associated with this approach:

1 - By pre-adjusting records instead of estimating HapAllele effects based on generalized least squares equations we ignore covariance structure and therefore bias the estimates downwards (Svishcheva et al., 2012).

2 - Each HapAllele being tested is also included in the kinship matrix, such that the HapAllele is included twice in the model: as fixed and random effect. This problem is known as proximal contamination (Listgarten et al., 2012).

In the first case, we can use genomic control to recover p-values to an unbiased scale (Devlin and Roeder, 1999; Amin et al., 2007). However, not much can be done regarding the estimates of the effects. As a general recommendation, if the user is only interested in p-values, the `ghap.assoc()` analysis should be sufficient. When effect estimates are of interest, the user can select genome-wide significant HapAlleles and include them as fixed effects in the full model in `ghap.mme()`. For the second case, a leave-one-chromosome-out (LOCO analysis) procedure can mitigate proximal contamination (Yang et al., 2014).

GWAS treating HapBlocks as fixed effects

GHap also supports HapBlock-based association. If the argument type in `ghap.assoc()` is supplied with the option "HapBlock" instead of the default "HapAllele", blocks are tested using the parameterization suggested by Da (2015). Briefly, for a given HapBlock, let HapAlleles 1, 2, ..., m have frequencies p_1, p_2,

$\ldots, p_m$. Also, let α_{1j} be the average effect of substituting HapAllele 1 by HapAllele j, such that there are $m-1$ independent substitution effects to be estimated. Under this framework, least squares estimates of substitution effects are obtained as:

$$\hat{\alpha} = (\mathbf{M}'\mathbf{M})^{-1}\mathbf{M}'\mathbf{y}_{\text{adj}}$$

where $\mathbf{M}$ is a n x $(m-1)$ centered HapGenotypes matrix for the HapAlleles within the block. Each entry m_{ij} of this matrix takes values $2p_j$, $-(1-2p_j)$ and $-2(1-p_j)$ for HapGenotypes 0, 1 and 2, respectively. If the HapAllele frequencies sum to 1 (i.e., no frequency prunning was used by the user during haplotyping), the `ghap.assoc()` takes HapAllele 1 to be the one presenting the smallest frequency within each block. Otherwise, the function assumes that the substitution effects of the pruned HapAlleles are known to be zero, and the contrast is carried out based on a phantom HapAllele 1.

Then, given the explained (ESS) and the residual (RSS) sum of squares, an F test is computed instead of the chi-squared test:

$$F = \frac{ESS/\nu_1}{RSS/\nu_2}$$
$$ESS = (\mathbf{M}\hat{\alpha} - \hat{\mu}_y)'(\mathbf{M}\hat{\alpha} - \hat{\mu}_y)$$
$$RSS = (\mathbf{y}_{\text{adj}} - \mathbf{M}\hat{\alpha})'(\mathbf{y}_{\text{adj}} - \mathbf{M}\hat{\alpha})$$
$$\hat{\mu}_y = \frac{1}{N}\sum_{i=1}^{n} y_i$$
$$\nu_1 = m - 1$$
$$\nu_2 = n - m - 1$$

Under the null hypothesis $VAR(\mathbf{M}\alpha) = 0$ (i.e, variance due to the HapBlock is zero) we have $F \sim F(\nu_1, \nu_2)$.

```
#HapBlock GWAS
gwas.block <- ghap.assoc(blmm = model, haplo = haplo, type = "HapBlock", ncores=2)

#Plot
gwas.block$POS <- (gwas.block$BP1+gwas.block$BP2)/2e+6
plot(gwas.block$POS,gwas.block$logP, xlab="Chromosome 2 position (in Mb)",
     ylab=expression(-log[10](p)), pch=20, col="#471FAA99")
abline(v=(haplo$bp1[1000]+haplo$bp2[1000])/2e+6,lty=3)
```

GWAS treating HapAlleles as random effects

Recall the linear mixed model:

$$\mathbf{y} = \mathbf{Xb} + \mathbf{Zu} + \mathbf{Zp} + \mathbf{e}$$

This model is equivalent to:

$$\mathbf{y} = \mathbf{Xb} + \mathbf{ZHa} + \mathbf{Zp} + \mathbf{e}$$

Assuming:

$$\mathbf{u} \mid \sigma^2_u \sim N(0, \mathbf{K}\sigma^2_u)$$

$$\mathbf{K} = q\mathbf{HDH}'$$

$$\mathbf{u} = \mathbf{Ha}$$

This means that we can convert between individual breeding values and HapAllele effects (Strandén and Garrick, 2009):

$$\hat{\mathbf{a}} = q\mathbf{DH}'\mathbf{K}^{-1}\hat{\mathbf{u}}$$

```
#BLUP solutions for HapAllele effects
blup <- ghap.blup(blmm = model, haplo = haplo, ncores=2)
blup$POS <- (blup$BP1+blup$BP2)/2e+6
plot(blup$POS,100*blup$pVAR, xlab="Chromosome 2 position (in Mb)",
     ylab="Haplotype variance explained (%)", pch=20, col="#471FAA99")
abline(v=(haplo$bp1[1000]+haplo$bp2[1000])/2e+6,lty=3)
```

Since we cannot test random effects in the same way we do for fixed effects, **GHap** offers a permutation test to assess significance of potential major HapAlleles. The permutation procedure consists in randomizing the vector of estimated breeding values and computing the null statistic $\max(\hat{\mathbf{a}})$. The permutation p-value is computed as the number of times the HapAllele effect was smaller than the null statistic.

```
#Permutation test
blup <- ghap.blup(blmm = model, haplo = haplo, ncores=2, nperm = 1000)
blup$POS <- (blup$BP1+blup$BP2)/2e+6
plot(blup$POS,-log10(blup$P), xlab="Chromosome 2 position (in Mb)",
     ylab=expression(-log[10](p)), pch=20, col="#471FAA99")
abline(v=(haplo$bp1[1000]+haplo$bp2[1000])/2e+6,lty=3)
abline(h=-log10(0.05))
```

It has been suggested that recallibrating estimates based on an iterative re-weighting procedure could aliviate the effects of shrinkage and improve estimation of random effects (Wang et al., 2012). Briefly, the method consists in computing HapAllele weights based on the BLUP solutions, computing a new weighted relationship matrix, fitting the linear mixed model with the new matrix, and converting the new estimated breeding values in updated BLUP solutions for HapAlleles. The example below shows that a single iteration in the simulated data can substantially improve the estimate of our major HapAllele effect. It has been suggested that 2 to 4 updates may be required.

```
#HapAllele weights
w <- nrow(blup)*blup$pVAR

#Weighted relationship matrix
K <- ghap.kinship(haplo, weights = w, batchsize = 100)

#Linear mixed model fitting
model <- ghap.mme(fixed = phenotype ~ 1, random = "individual", data = sim$data, K = K
                  varcomp = c(model$vare,model$varu))

#Re-weighted BLUP solutions
blup <- ghap.blup(blmm = model, haplo = haplo, weights = w, ncores=2)

#Plot
blup$POS <- (blup$BP1+blup$BP2)/2e+6
plot(blup$POS,100*blup$pVAR, xlab="Chromosome 2 position (in Mb)",
```

```
     ylab="Haplotype variance explained (%)", pch=20, col="#471FAA99")
abline(v=(haplo$bp1[1000]+haplo$bp2[1000])/2e+6,lty=3)

#Estimated x Simulated
sim$major.effect
blup$SCORE[1000]
```

HapAllele profiling

The profile for each individual is calculated as:

$$\sum_{i=1}^{m}(h_i a_i)$$

where relative to HapAllele i, h_i is the number of copies and a_i is a user-defined score. By default, if scores are provided for only a subset of the HapAlleles, the missing alleles scores will be set to zero. This function has the same spirit as the profiling routine implemented in the score option in PLINK (Purcell et al., 2007; Chang et al., 2015). This function can be useful for analyses involving cross-validation of genomic predictions based on BLUP solutions of HapAllele effects or scoring admixture proportions from the output of `ghap.ancestral()`. Below is an example using simulated scores from a normal distribution:

```
# Create a score data.frame
score <- NULL
score$BLOCK <- haplo$block
score$CHR <- haplo$chr
score$BP1 <- haplo$bp1
score$BP2 <- haplo$bp2
score$ALLELE <- haplo$allele
set.seed(1988)
score$SCORE <- rnorm(length(score$ALLELE))
score <- data.frame(score,stringsAsFactors = FALSE)

# Compute profiles
profile <- ghap.profile(score, haplo, ncores = 2)
head(profile)
#     POP      ID    PROFILE
#   1 ASW NA19904 -38.410381
#   2 ASW NA20340 -12.250027
#   3 ASW NA20297 -45.473774
#   4 ASW NA20281  -7.360974
#   5 ASW NA20348 -36.271198
#   6 ASW NA20300  40.912226
```

Using GHap outputs in third-party software

When the haplotyping procedure is performed using very large datasets, post hoc analyses may be too computationally demanding to be performed in **R**. Also, existing pipelines designed to analyze bi-allelic SNP data can be extended to the analysis of haplotypes by simply incorporating the output generated by the `ghap.hap2tped()` function in **GHap**. This function creates a set of files that mimic a standard PLINK (Purcell et al., 2007; Chang et al., 2015) tped file, where HapAllele counts 0, 1 and 2 are recoded as NN, NH and HH genotypes (N = NULL and H = haplotype allele), as if HapAlleles were bi-alelic markers. This codification is acceptable for any given analysis relying on genotype counts, as long as the user specifies that

the analysis should be done using the H allele as reference for counts. You can specify reference alleles using the .tref file in PLINK with the reference-allele command.The name for each pseudo-marker is composed by a concatenation (separated by "_") of block name, start, end, and haplotype allele identity. Pseudo-marker positions are computed as (start+end)/2.

The following lines of code show one example of how the output from **GHap** can be articulated with analyses that are routinely applied to unphased SNP marker data. First, we can export the **GHap.haplo** object to use in PLINK:

```
# Convert to tped
ghap.hap2tped(infile = "example", outfile = "example")
#   Creating .tfam file...Done.
#   Creating .tref file...Done.
#   Creating .tped file for 1750 haplotype alleles.
#   1750 haplotype alleles converted.
```

Then, we can use PLINK to perform a principal components analysis on our data:

```
#Converting the tped output to PLINK binary
plink --tfile example --reference-allele example.tref --make-bed --out example

#Performing PCA analysis in PLINK
#Correlations and scale with the GHap package are almost perfect (r = 0.999)
plink --bfile example --reference-allele example.tref --pca 2 --out example
```

Notes on handling multiple chromosomes and analysis of single marker data

By default **GHap** works at single chromosome data, specially when it comes to the haplotyping procedure. However, once HapAlleles have been called on each chromosome, the user can choose to load one chromosome at a time or to load all chromosomes together with *ghap.loadhaplo()*. This can be typically achieved by the use of the *ghap.mergehaplo()* function, or alternatively by concatenation of single chromosome files.

You can also fool **GHap** to take in single SNP data (say you wish to compare haplotype x single SNP association results). To do so, you only need to see SNPs as distinct 1bp HapBlocks, and count number of copies of a particular allele (e.g., minor or reference allele). Then, *ghap.loadhaplo()* will naturally load single SNP data. Be aware that for some analyses special consideration may be required, so be sure that you know exactly what you are doing!

Benchmarking

Benchmarking of the main tasks in the package was first performed in a Dell PowerEdge-T410 workstation with 16 GB RAM and two 64-bit Intel Xeon 2.13 GHz CPUs, running R v3.2.5 under Ubuntu 10.04 LTS. Performance was evaluated using the HapMap data with varying number of cores.

Benchmarking of GHap v1.2.0 with varying numbers of cores.
1,011 samples and 20,000 markers were used.
Time was measured in seconds and averaged over 10 replicates.

Task	Number of cores			
	1	2	4	8
Load data	9.86±0.11	-	-	-
Filtering*	8.20±0.23	4.20±0.14	2.60±0.13	1.59±0.04
Haplotyping	169.50±1.01	82.47±0.22	42.00±0.33	23.00±0.19

*Minor allele frequency prunning and subsetting

Another benchmarking was conducted to assess the influence of dataset size on performance. This benchmarking was done using a Dell T5500 workstation with 24 GB and 64-bit Intel Xeon 3.07GHz CPU, running R v3.2.3 under Red Hat Enterprise Linux Workstation release 6.7.

Benchmarking of GHap v1.2.0 with 8 cores and varying numbers of markers and subjects.
Time was measured in seconds and averaged over 10 replicates.

Markers	Samples	Task		
		Load data	Filtering	Haplotyping
10,000	1,000	2.62±0.01	0.34±0.04	6.29±0.10
	5,000	13.14±0.03	0.91±0.01	28.33±0.25
	10,000	26.04±0.07	1.49±0.05	57.42±0.41
	50,000	134.44±0.54	7.98±0.20	282.44±1.43
100,000	1,000	28.02±0.07	3.30±0.09	76.64±0.31
	5,000	143.01±0.22	9.95±0.13	286.89±0.90
	10,000	281.83±0.91	26.73±0.51	561.97±1.48
	50,000*	-	-	-

*The analysis of 50,000 samples and 100,000 markers consumed more than
the maximum RAM available (24GB) and was unfeasible using available hardware

In summary, **GHap** scales linearly as a function of markers or samples. We noticed a limitation in the analysis of a large number of individuals, but this was related to RAM availability. Analyses of such large datasets may be accomplished using high-performance computing facilities or by subdividing the data in batches with smaller sample sizes. If no HapAllele frequency filter is determined, batching by sample will not affect haplotyping since HapGenotypes are called individually.

References

N. Amin et al. A Genomic Background Based Method for Association Analysis in Related Individuals. PLoS ONE. 2007. 2:e1274.

S. Bolormaa et al. Detection of chromosome segments of zebu and taurine origin and their effect on beef production and growth. J. Anim. Sci. 2011. 89:2050-2060.

C. C. Chang et al. Second-generation PLINK: rising to the challenge of larger and richer datasets. Gigascience. 2015. 4, 7.

Y. Da. Multi-allelic haplotype model based on genetic partition for genomic prediction and variance component estimation using SNP markers. BMC Genet. 2015. 16:144.

B. Devlin and K. Roeder. Genomic control for association studies. Biometrics. 1999. 55:997-1004.

J. Listgarten et al. Improved linear mixed models for genome-wide association studies. Nat. Methods. 2012. 9:525-526.

S. B. Gabriel et al. The structure of haplotype blocks in the human genome. Science. 2002. 296:2225-2229.

C. R. Henderson. Sire evaluation and genetic trends. Journal of Animal Science. 1973. Symposium: 10-41.

G. N. Norén et al. Shrinkage observed-to-expected ratios for robust and transparent large-scale pattern discovery. Stat Methods Med Res. 2013. 22,57-69.

J. O'Connell et al. A general approach for haplotype phasing across the full spectrum of relatedness. PLOS Genetics. PLOS Genet. 2014. 10:e1004234.

P. Pérez and G. de Los Campos. Genome-Wide Regression and Prediction with the BGLR Statistical Package. Genetics. 2014. 198:483-495.

S. Purcell et al. PLINK: a tool set for whole-genome association and population-based linkage analyses. Am. J. Hum. Genet. 2007. 81, 559-575.

D. A. Sorensen et al. Bayesian inference in threshold models using Gibbs sampling. Genet Sel Evol. 1995. 27:229-249.

I. Strandén and D.J. Garrick. Technical note: derivation of equivalent computing algorithms for genomic predictions and reliabilities of animal merit. J Dairy Sci. 2009. 92:2971-2975.

G. R. Svishcheva et al. Rapid variance components-based method for whole-genome association analysis. Nat Genet. 2012. 44:1166-1170.

The International HapMap 3 Consortium. Integrating common and rare genetic variation in diverse human populations. Nature. 2010. 467, 52-58.

P. M. VanRaden. Efficient methods to compute genomic predictions. J. Dairy. Sci. 2008. 91:4414-4423.

C. S. Wang et al. Bayesian analysis of mixed linear models via Gibbs sampling with an application to litter size in Iberian pigs. Genet Sel Evol. 1994. 26:91-115.

H. Wang et al. Genome-wide association mapping including phenotypes from relatives without genotypes. Genet Res. 2012. 94:73-83.

J. Yang et al. Advantages and pitfalls in the application of mixed-model association methods. Nat. Genet. 2014. 46: 100-106.

Appendix 1 - Frequentist linear mixed model

General formulation

Consider the model:

$$\mathbf{y} = \mathbf{Xb} + \mathbf{Zu} + \mathbf{Zp} + \mathbf{e}$$

with likelihood:

$$\mathbf{y} \mid \mathbf{b}, \mathbf{u}, \mathbf{p}, \sigma^2_u, \sigma^2_p, \sigma^2_e \sim N(\mathbf{Xb}, \mathbf{V})$$

and variance-covariance matrix:

$$\mathbf{V} = \mathbf{ZKZ}'\sigma^2_u + \mathbf{ZZ}'\sigma^2_p + \mathbf{I}\sigma^2_e$$

Likelihood

The log-likelihood function is defined as:

$$\log[Pr(\mathbf{y} \mid \mathbf{b}, \mathbf{u}, \mathbf{p}, \sigma^2_u, \sigma^2_p, \sigma^2_e)] = -\frac{1}{2}\left[n\log(2\pi) + \log|\mathbf{V}| + (\mathbf{y}-\mathbf{Xb})'\mathbf{V}^{-1}(\mathbf{y}-\mathbf{Xb})\right]$$

It is useful to define:

$$\begin{aligned}
\mathbf{V} &= \mathbf{UDU}' \\
\mathbf{V}^{-1} &= \mathbf{UD}^{-1}\mathbf{U}' \\
\mathbf{U} = \text{eigvec}(\mathbf{V}) &= \text{eigvec}(\mathbf{ZKZ}' + \mathbf{ZZ}' + \mathbf{I}) \\
\mathbf{D} &= \text{diag}(\mathbf{d}) \\
\mathbf{d} = \text{eigval}(\mathbf{V}) &= \text{eigval}(\mathbf{ZKZ}')\sigma^2_u + \text{eigval}(\mathbf{ZZ}')\sigma^2_p + \text{eigval}(\mathbf{I})\sigma^2_e \\
&= \mathbf{a}\sigma^2_u + \mathbf{b}\sigma^2_p + \mathbf{1}\sigma^2_e \\
|\mathbf{V}| &= \prod_{i=1}^{n} d_i
\end{aligned}$$

And re-write the log-likelihood as:

$$\log[Pr(\mathbf{y} \mid \mathbf{b}, \mathbf{u}, \mathbf{p}, \sigma^2_u, \sigma^2_p, \sigma^2_e)] \propto -\frac{1}{2}\left[\sum_{i=1}^{n}\log(d_i) + \sum_{i=1}^{n}\frac{q_i^2}{d_i}\right]$$

where $\mathbf{q} = \mathbf{U}'(\mathbf{y}-\mathbf{Xb})$. With this formulation, we avoid expensive matrix inversions by computing $\mathbf{U}$, $\mathbf{a}$ and $\mathbf{b}$ only once. The negative log-likelihood is minimized using the L-BFGS-B method implemented in the *optim()* function in the **stats** package. Estimates of variance components from the minimization procedure are then used in Henderson's mixed model equations to find BLUE and BLUP solutions for fixed and random effects.

Appendix 2 - Bayesian linear mixed model

General formulation

Consider the model:

$$\mathbf{y} = \mathbf{Xb} + \mathbf{Zu} + \mathbf{e}$$

with likelihood:

$$y_i \mid \mathbf{b}, \mathbf{u}, \sigma_u^2, \sigma_e^2 \sim N(\eta_i, \sigma_e^2)$$
$$\eta_i = \sum_{j=1}^{p} x_{ij} b_j + \sum_{k=1}^{m} z_{ik} u_k$$

And prior densities for the parameters:

$$b_j \propto \text{constant}$$
$$u_k \mid \sigma_u^2 \sim N(0, \sigma_u^2)$$
$$\sigma_u^2 \sim \chi^{-2}(\nu_u, S_u^2)$$
$$\sigma_e^2 \sim \chi^{-2}(\nu_e, S_e^2)$$

Also recall the normal and scaled inverse chi-squared probability density functions:

$$X \sim N(\mu, \sigma^2),\ Pr(X = x \mid \mu, \sigma^2) = (2\pi\sigma^2)^{-1/2} \exp\left\{ -\frac{(x-\mu)^2}{2\sigma^2} \right\}$$

$$X \sim \chi^{-2}(\nu, S^2),\ Pr(X = x \mid \nu, S^2) = \frac{(S^2\nu/2)^{(\nu/2)}}{\Gamma(\nu/2)} x^{-(\nu+2)/2} \exp\left\{ -\frac{\nu S^2}{2x^2} \right\}$$

Likelihood

For a single observation y_i we have:

$$Pr(y_i \mid \mathbf{b}, \mathbf{u}, \sigma_u^2, \sigma_e^2) = (2\pi\sigma_e^2)^{-1/2} \exp\left\{ -\frac{(y_i - \eta_i)^2}{2\sigma_e^2} \right\}$$

And for all n observations:

$$Pr(\mathbf{y} \mid \mathbf{b}, \mathbf{u}, \sigma_u^2, \sigma_e^2) = \prod_{i=1}^{n} Pr(y_i \mid \mathbf{b}, \mathbf{u}, \sigma_u^2, \sigma_e^2) = \prod_{i=1}^{n} (2\pi\sigma_e^2)^{-1/2} \exp\left\{ -\frac{(y_i - \eta_i)^2}{2\sigma_e^2} \right\}$$

$$= (2\pi\sigma_e^2)^{-n/2} \exp\left\{ -\frac{\sum_{i=1}^{n} (y_i - \eta_i)^2}{2\sigma_e^2} \right\}$$

Prior for random effects

For a single random effect u_k we have:

$$Pr(u_k \mid \sigma_u^2) = (2\pi\sigma_u^2)^{-1/2}\exp\left\{ -\frac{u_k^2}{2\sigma_u^2} \right\}$$

And for all m random effects:

$$Pr(\mathbf{u} \mid \sigma_u^2) = \prod_{k=1}^{m} p(u_k \mid \sigma_u^2) = \prod_{k=1}^{m} (2\pi\sigma_u^2)^{-1/2}\exp\left\{ -\frac{u_k^2}{2\sigma_u^2} \right\}$$

$$= (2\pi\sigma_u^2)^{-m/2}\exp\left\{ -\frac{\sum_{k=1}^{m} u_k^2}{2\sigma_u^2} \right\}$$

Prior for variance of random effects

$$Pr(\sigma_u^2) = \frac{(S_u^2\nu_u/2)^{(\nu_u/2)}}{\Gamma(\nu_u/2)}(\sigma_u^2)^{-(\nu_u+2)/2}\exp\left\{ -\frac{\nu_u S_u^2}{2\sigma_u^2} \right\}$$

Prior for residual variance

$$Pr(\sigma_e^2) = \frac{(S_e^2\nu_e/2)^{(\nu_e/2)}}{\Gamma(\nu_e/2)}(\sigma_e^2)^{-(\nu_e+2)/2}\exp\left\{ -\frac{\nu_e S_e^2}{2\sigma_e^2} \right\}$$

Full and conditional posterior distributions

Here we simply present the posterior distributions, which are derived in greater detail later.

$$Pr(\mathbf{b}, \mathbf{u}, \sigma_u^2, \sigma_e^2 \mid \mathbf{y}) \propto Pr(\mathbf{y} \mid \mathbf{b}, \mathbf{u}, \sigma_u^2, \sigma_e^2) Pr(\mathbf{u} \mid \sigma_u^2) Pr(\sigma_u^2) Pr(\sigma_e^2)$$

$$Pr(b_j \mid \text{ELSE}) \propto N\left(\frac{\sum_{i=1}^{n} x_{ij}(y_i - \eta_{ij})}{\sum_{i=1}^{n} x_{ij}^2}, \frac{\sigma_e^2}{\sum_{i=1}^{n} x_{ij}^2} \right)$$

$$Pr(u_k \mid \text{ELSE}) \propto N\left(\frac{\sum_{i=1}^{n} z_{ik}(y_i - \eta_{ik})}{\left(\sum_{i=1}^{n} z_{ik}^2\right) + \frac{\sigma_e^2}{\sigma_u^2}}, \frac{\sigma_e^2}{\left(\sum_{i=1}^{n} z_{ik}^2\right) + \frac{\sigma_e^2}{\sigma_u^2}} \right)$$

$$Pr(\sigma_u^2 \mid \text{ELSE}) \propto \chi^{-2}\left(m + \nu_u, \frac{\sum_{k=1}^{m} u_k^2 + \nu_u S_u^2}{m + \nu_u} \right)$$

$$Pr(\sigma_e^2 \mid \text{ELSE}) \propto \chi^{-2}\left(n + \nu_u, \frac{\sum_{i=1}^{n} (y_i - \eta_i)^2 + \nu_e S_e^2}{n + \nu_e} \right)$$

where η_{ij} is simply η_i computed based on all effects except b_j, and η_{ik} is simply η_i computed based on all effects except u_k. As the expected value of a scaled inverse chi-squared distribution is $E[X] = \frac{\nu S^2}{\nu - 2}$, we define $S_u^2 = \frac{\sigma_u^2(\nu_u - 2)}{\nu_u}$ for an arbitrarily value of ν_u. Likewise, $S_e^2 = \frac{\sigma_e^2(\nu_e - 2)}{\nu_e}$ for an arbitrarily value of ν_e.

Gibbs sampler

For each parameter, the algorithm works as follows:

1 - At iteration $i = 0$ initialize all parameters with a starting guess.

2 - For each parameter, take a sample from the respective conditional posterior distribution and add to the cumulative posterior estimates.

3 - Increment $i = i + 1$.

4 - Repeat 2 and 3 until the maximum number of simulations is reached.

5 - Once the maximum number of simulations is reached, divide the cumulative posterior estimates by the effective number of simulations (number of simulations divided by the thinning interval).

An initial guess for the variance components can be given by the user through the heritability argument. Starting guesses for fixed effects are taken from ordinary least squares estimates, whereas all random effects start with value zero. At every iteration, parameters are sampled in the following order: liabilities, thresholds, fixed effects, random effects and variance components. During burn-in, samples are not added to the cumulative posterior estimates and the iteration number is not incremented. Similarly, samples within thinning intervals are not added to the cumulative posterior estimates.

Model with permanent environmental effects and correlated random effects

The conditional posterior distribution of permanent environmental effects is identical to the uncorrelated effects discussed in this appendix. We can also incorporate covariance of random effects into the model without effectively changing the full and conditional posterior distributions. The targeted model assumes:

$$\begin{aligned}
\mathbf{y} &= \mathbf{Xb} + \mathbf{Zu} + \mathbf{Zp} + \mathbf{e} \\
\mathbf{u} &\sim N(0, \mathbf{K}\sigma_u^2) \\
\mathbf{p} &\sim N(0, \mathbf{I}\sigma_p^2) \\
\mathbf{e} &\sim N(0, \mathbf{W}\sigma_e^2)
\end{aligned}$$

If we define:

$$\begin{aligned}
\mathbf{K} &= \mathbf{LL}' \\
\mathbf{Z}^* &= \mathbf{ZL} \\
\mathbf{u}^* &= \mathbf{L}^{-1}\mathbf{u}
\end{aligned}$$

then we can fit the equivalent model

$$\mathbf{y} = \mathbf{Xb} + \mathbf{Z}^*\mathbf{u}^* + \mathbf{Zp} + \mathbf{e}$$

which assumes

$$\mathbf{u}^* \sim N(0, \mathbf{I}\sigma_u^2)$$

Once the model is fitted, we can solve for $\hat{\mathbf{u}} = \mathbf{L}\hat{\mathbf{u}}^*$.

Fitting categorical data

Let the response variable be an ordered categorical variable such that $\mathbf{y} \in \{1, 2, ..., K\}$. We assume categories represent classifications imposed by thresholds γ_k of an unobserved Gaussian random variable $\mathbf{z}$ ("liability"), such that:

$$y_i = k, \text{ if } \gamma_k \leq z_i \leq \gamma_{k+1}$$

We assume $\gamma_1 = -\infty$ and $\gamma_{K+1} = \infty$. At each Monte Carlo simulation, z_i is sampled from a normal distribution with mean η_i and variance 1, truncated at γ_k and γ_{k+1}. Notice that, in the way we have defined

the likelihood of the data, this is equivalent to setting the residual variance to one. Then, the sampling algorithm continues as z_i was the actual response variable.

Derivation for the conditional posterior of fixed effects

We start by marginalizing b_j from the likelihood function. This is achieved by computing η_i based on all effects except the targeted effect b_j or equivalently:

$\eta_{ij} = \eta_i - x_{ij}b_j$

Then we define:

$y_{ij} = y_i - \eta_{ij}$

Such that:

$$p(b_j \mid \text{ELSE}) \propto (2\pi\sigma_e^2)^{-n/2}\exp\left\{ - \frac{\sum_{i=1}^{n}(y_{ij}-x_{ij}b_j)^2}{2\sigma_e^2} \right\}$$

Now, we will try to keep on dropping all terms that do not depend on b_j. We can start by dropping $(2\pi\sigma_e^2)^{-n/2}$ and express the quadratic term in vector form:

$\sum_{i=1}^{n}(y_{ij} - x_{ij}b_j)^2 = (\mathbf{y}_j - \mathbf{x}_j b_j)'(\mathbf{y}_j - \mathbf{x}_j b_j)$

we can now expand the quadratic to obtain

$\mathbf{y}_j{}'\mathbf{y}_j - 2\mathbf{x}_j{}'\mathbf{y}_j b_j + \mathbf{x}_j{}'\mathbf{x}_j b_j^2$

Since $\mathbf{y}_j{}'\mathbf{y}_j$ does not depend on b_j, we can drop it without compromising the proportionality. If we factor $\mathbf{x}_j{}'\mathbf{x}_j$ out we obtain:

$\mathbf{x}_j{}'\mathbf{x}_j[b_j^2 - 2(\mathbf{x}_j{}'\mathbf{x}_j)^{-1}\mathbf{x}_j{}'\mathbf{y}_j b_j]$

Let $a_j = (\mathbf{x}_j{}'\mathbf{x}_j)^{-1}\mathbf{x}_j{}'\mathbf{y}_j$. Now we have:

$\mathbf{x}_j{}'\mathbf{x}_j[b_j^2 - 2a_j b_j]$

It seems $b_j^2 - 2a_j b_j$ is a quadratic expansion of $(b_j - a_j)^2$ missing the a_j^2 term, that is:

$\mathbf{x}_j{}'\mathbf{x}_j[b_j^2 - 2a_j b_j + a_j^2 - a_j^2]$

Therefore, we can now write:

$\mathbf{x}_j{}'\mathbf{x}_j[(b_j - a_j)^2 - a_j^2]$

Again, a_j^2 does not depend on b_j so we can drop it:

$\mathbf{x}_j{}'\mathbf{x}_j[(b_j - a_j)^2]$

Back to the full expression:

$$p(b_j \mid \text{ELSE}) \propto \exp\left\{ - \frac{\mathbf{x}_j{}'\mathbf{x}_j[(b_j-a_j)^2]}{2\sigma_e^2} \right\}$$

We can conveniently write $v_j^2 = (\mathbf{x}_j{}'\mathbf{x}_j)^{-1}\sigma_e^2$ to obtain:

$$p(b_j \mid \text{ELSE}) \propto \exp\left\{ - \frac{(b_j-a_j)^2}{2v_j^2} \right\}$$

Therefore, the conditional posterior distribution for b_j is proportional to:

$N(a_j, v_j^2)$

where

$$a_j = (\mathbf{x}_j{}'\mathbf{x}_j)^{-1}\mathbf{x}_j{}'\mathbf{y}_j = \frac{\sum_{i=1}^{n} x_{ij}y_{ij}}{\sum_{i=1}^{n} x_{ij}^2}$$

$v_j^2 = (\mathbf{x}_j{}'\mathbf{x}_j)^{-1}\sigma_e^2 = \frac{\sigma_e^2}{\sum_{i=1}^n x_{ij}^2}$

Derivation for the conditional posterior of random effects

The rationale here is similar to that from the fixed effects. We marginalize u_k from the likelihood function by computing η_i based on all effects except the targeted effect u_k. This is equivalent to computing:

$\eta_{ik} = \eta_k - z_{ik}u_k$

Then we define:

$y_{ik} = y_i - \eta_{ik}$

Such that:

$$p(u_k \mid \text{ELSE}) \propto (2\pi\sigma_e^2)^{-n/2}\exp\left\{ - \frac{\sum_{i=1}^n (y_{ik} - z_{ik}u_k)^2}{2\sigma_e^2} \right\} \times (2\pi\sigma_u^2)^{-1/2}\exp\left\{ - \frac{u_k^2}{2\sigma_u^2} \right\}$$

We can drop $(2\pi\sigma_e^2)^{-n/2}$ and $(2\pi\sigma_u^2)^{-n/2}$ rearrange the expression:

$$p(u_k \mid \text{ELSE}) \propto \exp\left\{ - \frac{\sum_{i=1}^n (y_{ik} - z_{ik}u_k)^2 + u_k^2\frac{\sigma_e^2}{\sigma_u^2}}{2\sigma_e^2} \right\}$$

If we perform similar manipulation of the quadratic term here as we did for the fixed effects we obtain:

$$p(u_k \mid \text{ELSE}) \propto \exp\left\{ - \frac{(u_k - a_k)^2}{2v_k^2} \right\}$$

Which is proportional to:

$N(a_k, v_k^2)$

where

$$a_k = (\mathbf{z}_k{}'\mathbf{z}_k + \frac{\sigma_e^2}{\sigma_u^2})^{-1}\mathbf{z}_k{}'\mathbf{y}_k = \frac{\sum_{i=1}^n z_{ik}y_{ik}}{\left(\sum_{i=1}^n z_{ik}^2\right) + \frac{\sigma_e^2}{\sigma_u^2}}$$

$$v_k^2 = (\mathbf{z}_j{}'\mathbf{z}_j + \frac{\sigma_e^2}{\sigma_u^2})^{-1}\sigma_e^2 = \frac{\sigma_e^2}{\left(\sum_{i=1}^n z_{ik}^2\right) + \frac{\sigma_e^2}{\sigma_u^2}}$$

Derivation for the conditional posterior of variance of random effects

The marginal posterior distribution for the variance of random effects is:

$$p(\sigma_u^2 \mid \text{ELSE}) \propto (2\pi\sigma_u^2)^{-m/2}\exp\left\{ - \frac{\sum_{k=1}^m u_k^2}{2\sigma_u^2} \right\} \times \frac{(S_u^2\nu/2)^{(\nu_u/2)}}{\Gamma(\nu_u/2)}(\sigma_u^2)^{-(\nu_u+2)/2}\exp\left\{ - \frac{\nu_u S_u^2}{2\sigma_u^2} \right\}$$

We can drop terms not depending on σ_u^2 and rearrange the expression to obtain:

$$p(\sigma_u^2 \mid \text{ELSE}) \propto (\sigma_u^2)^{-(m+\nu_u+2)/2}\exp\left\{ - \frac{\sum_{k=1}^m u_k^2 + \nu_u S_u^2}{2\sigma_u^2} \right\}$$

If we let $d_u = m + \nu_u$ and $s_u^2 = (\sum_{k=1}^m u_k^2 + \nu_u S_u^2)/d_u$, then:

$$p(\sigma_u^2 \mid \text{ELSE}) \propto (\sigma_u^2)^{-(d_u+2)/2}\exp\left\{ - \frac{d_u s_u^2}{2\sigma_u^2} \right\}$$

Which is proportional to:

$\chi^{-2}(d_u, s_u^2)$

Derivation for the conditional posterior of residual variance

The marginal posterior distribution for the residual variance is:

$$p(\sigma_u^2 \mid \text{ELSE}) \propto (2\pi\sigma_e^2)^{-n/2}\text{exp}\left\{ -\frac{\sum_{i=1}^{n}(y_i-\mu_i)^2}{2\sigma_e^2} \right\} \times \frac{(S_e^2\nu_e/2)^{(\nu_e/2)}}{\Gamma(\nu_e/2)}(\sigma_e^2)^{-(\nu_e+2)/2}\text{exp}\left\{ -\frac{\nu_e S_e^2}{2\sigma_e^2} \right\}$$

We can follow the case of the variance of the random effects and conduct similar term dropping and expression rearrangements:

$$p(\sigma_e^2 \mid \text{ELSE}) \propto (\sigma_e^2)^{-(n+\nu_u+2)/2}\text{exp}\left\{ -\frac{\sum_{i=1}^{n}(y_i-\eta_i)^2+\nu_e S_e^2}{2\sigma_e^2} \right\}$$

If we let $d_e = m + \nu_e$ and $s_e^2 = (\sum_{i=1}^{n}(y_i - \eta_i)^2 + \nu_e S_e^2)/d_e$, then:

$$p(\sigma_e^2 \mid \text{ELSE}) \propto (\sigma_e^2)^{-(d_e+2)/2}\text{exp}\left\{ -\frac{d_e s_e^2}{2\sigma_e^2} \right\}$$

Which is proportional to:

$\chi^{-2}(d_e, s_e^2)$

Appendix 3 - Model with heteroskedastic residuals

We can incorporate heteroskedastic residuals into the model without effectively changing the fitting algorithms. The targeted model assumes:

$$\begin{gathered}\mathbf{y} = \mathbf{Xb} + \mathbf{Zu} + \mathbf{Zp} + \mathbf{e} \\ \mathbf{u} \sim N(0, \mathbf{K}\sigma_u^2) \\ \mathbf{p} \sim N(0, \mathbf{I}\sigma_p^2) \\ \mathbf{e} \sim N(0, \mathbf{W}\sigma_e^2) \\ \mathbf{W} = \text{diag}(w_i)\end{gathered}$$

If we define:

$$\begin{gathered}\mathbf{W}^{-1} = \mathbf{W}^{-1/2}\mathbf{W}^{-1/2} \\ \mathbf{W}^{-1/2} = \text{diag}\left(1/\sqrt{w_i}\right) \\ \mathbf{y}^* = \mathbf{W}^{-1/2}\mathbf{y} \\ \mathbf{X}^* = \mathbf{W}^{-1/2}\mathbf{X} \\ \mathbf{Z}^* = \mathbf{W}^{-1/2}\mathbf{Z} \\ \mathbf{e}^* = \mathbf{W}^{-1/2}\mathbf{e}\end{gathered}$$

then we can fit the equivalent model

$$\mathbf{y}^* = \mathbf{X}^*\mathbf{b} + \mathbf{Z}^*\mathbf{u} + \mathbf{Z}^*\mathbf{p} + \mathbf{e}^*$$

which assumes

$$\mathbf{e}^* \sim N(0, \mathbf{I}\sigma_e^2)$$

Once the model is fitted, we can solve for $\hat{\mathbf{e}} = \mathbf{W}^{1/2}\hat{\mathbf{e}}^*$.