LEA: An R Package for Landscape and Ecological Association Studies

Eric Frichot and Olivier François
Université Grenoble-Alpes,
Centre National de la Recherche Scientifique,
TIMC-IMAG UMR 5525, Grenoble, 38042, France.

Contents

1 Overview **1**

2 Introduction **2**
2.1 Input files 3

3 Analysis of population structure and imputation of missing data **3**
3.1 Principal Component Analysis 4
3.2 Inference of individual admixture coefficients using snmf . . . 5
3.3 Population differentation tests using snmf 8
3.4 Missing genotype imputation using snmf 9

4 Ecological associations tests using lfmm **10**

1 Overview

LEA is an R package dedicated to landscape genomics and ecological association tests (Frichot and Francois, 2015). LEA can run analyses of population structure and genomewide tests for local adaptation. The package includes statistical methods for estimating ancestry coefficients from large genotypic matrices and for evaluating the number of ancestral populations (snmf, pca). It performs statistical tests using latent factor mixed models for identifying genetic polymorphisms that exhibit high correlation with environmental gradients (lfmm). LEA is mainly based on optimized C programs that can scale with the dimension of large data sets.

2 Introduction

The goal of this tutorial is to give an overview of the main functionalities of the R package LEA. It will show the main steps of analysis, including 1) analysing population structure and preparing a genotypic matrix for genomewide association studies, 2) fitting GWAS latent factor mixed models to the data and extracting candidate regions of interest.

As some functions may take a few hours to analyse very large datasets, output files are written into text files that can be read by LEA after each batch of runs (called a 'project'). We advise creating working directories containing genotypic data and covariables when starting LEA. Note that two files with the same names but a different extension are assumed to contain the same data in distinct formats.

```
# creation of a directory for LEA analyses
dir.create("LEA_analyses")

## Warning in dir.create("LEA_analyses"):  'LEA_analyses' existe
dj

# set the created directory as the working directory
setwd("LEA_analyses")
```

This tutorial is based on a small dataset consisting of 400 SNPs genotyped for 50 diploids individuals. The last 50 SNPs are correlated with an environmental variable, and represent the target loci for an association analysis. Similar artificial data were analyzed in the computer note introducing the R package LEA (Frichot and Francois, 2015).

```
library(LEA)
# Creation a the genotypic file: "genotypes.lfmm"
# The data include 400 SNPs for 50 individuals.
data("tutorial")
# Write genotypes in the lfmm format
write.lfmm(tutorial.R, "genotypes.lfmm")
# Write genotypes in the geno format
write.geno(tutorial.R, "genotypes.geno")
# creation of an environment gradient file: gradient.env.
# The .env file contains a single ecological variable
# for each individual.
write.env(tutorial.C, "gradients.env")
```

Note that the LEA package is to be able to handle very large population genetic data sets. Genomic data are processed using fast C codes wrapped into the R code. Most LEA functions use character strings containing paths to input files as arguments.

2.1 Input files

The R package LEA can handle several input file formats for genotypic matrices. More specifically, the package uses the lfmm and geno formats, and provides functions to convert from other formats such as ped, vcf, and ancestrymap formats. The program VCFTOOLS can be very useful in providing one of those format (ped is the closest to an lfmm matrix).

The lfmm and geno formats can also be used for coding multiallelic marker data (eg, microsatellites). For multiallelic marker data, the conversion functionstruct2geno() converts files in the STRUCTURE format in the geno or lfmm formats. LEA can also process allele frequency data if they are encoded in the lfmm format. In that case, the lfmm function will use allele counts for populations in its model.

Phenotypic traits and ecological predictors must be formatted in the env format. This format corresponds to a matrix where each variable is represented as a column (Frichot et al., 2013). It uses the .env extension.

When using ecological data, we often need to decide which variables should be used among a large number of ecological indicators (eg, climatic variables), we suggest that users summarize their data using linear combinations of those indicators. Considering principal component analysis and using the first principal components as proxies for ecological gradients linked to selective forces can be useful in this context.

The LEA package can handle missing data in population structure analyses. In association analyses, missing genotypes must be replaced by imputed values using a missing data imputation method. We encourage users to remove their missing data by using the function impute(), which is based on population structure analysis(see next section). Note that specialized genotype imputation programs such as BEAGLE, IMPUTE2 or MENDEL-IMPUTE could provide better imputation results than LEA. Filtering out rare variants – retaining minor allele frequency greater than 5 percent –, and removing regions in strong LD may also result in better analyses with LEA.

3 Analysis of population structure and imputation of missing data

The R package LEA implements two classical approaches for the estimation of population genetic structure: principal component analysis (pca) and admixture analysis (Patterson et al., 2006; Pritchard et al., 2000) using sparse nonnegative matrix factorization (snmf). The algorithms programmed in

LEA are improved versions of pca and admixture analysis, and they are able to process large genotypic matrices efficiently.

3.1 Principal Component Analysis

The LEA function `pca()` computes the scores of a pca for a genotypic matrix, and returns a screeplot for the eigenvalues of the sample covariance matrix. Using `pca`, an object of class `pcaProject` is created. This object contains a path to the files storing eigenvectors, eigenvalues and projections.

```
# run of pca
# Available options, K (the number of PCs),
#                    center and scale.
# Create files: genotypes.eigenvalues - eigenvalues,
#               genotypes.eigenvectors - eigenvectors,
#               genotypes.sdev - standard deviations,
#               genotypes.projections - projections,
# Create a pcaProject object: pc.
pc = pca("genotypes.lfmm", scale = TRUE)
```

The number of "significant" components can be evaluated using graphical methods based on the screeplot (Figure 1). The knee in the screeplot indicates that there are around $K = 4$ major components in the data (≈ 5 genetic clusters). Following (Patterson et al., 2006), the `tracy.widom` function computes Tracy-Widom tests for each eigenvalue as follows.

```
# Perfom Tracy-Widom tests on all eigenvalues.
# create file: tuto.tracyWidom - tracy-widom test information.
tw = tracy.widom(pc)
```

```
# display p-values for the Tracy-Widom tests (first 5 pcs).
tw$pvalues[1:5]

## [1] 8.000e-09 8.000e-09 8.000e-09 1.503e-04 3.152e-02
```

```
# plot the percentage of variance explained by each component
plot(tw$percentage)
```

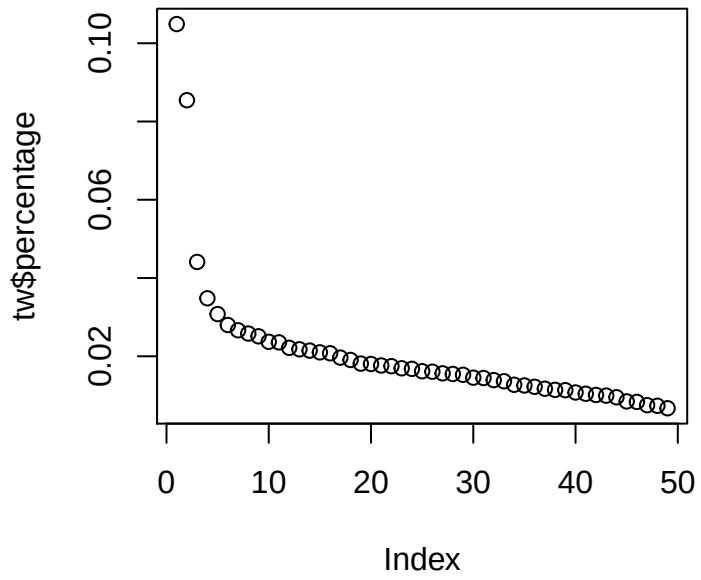

Figure 1: Screeplot for the percentage of variance explained by each component in a PCA of the genetic data. The knee at $K = 4$ indicates that there are 5 major genetic clusters in the data.

3.2 Inference of individual admixture coefficients using `snmf`

`LEA` includes the `R` function **`snmf`** that estimates individual admixture coefficients from the genotypic matrix with provides results very close to Bayesian clustering programs (Pritchard et al., 2000; François and Durand, 2010). Assuming K ancestral populations, the `R` function **`snmf`** provides least-squares estimates of ancestry proportions (Frichot et al., 2014).

```
# main options
# K = number of ancestral populations
# entropy = TRUE: computes the cross-entropy criterion,
# CPU = 4 the number of CPUs.
project = NULL
project = snmf("genotypes.geno",
               K = 1:10,
```

```
                entropy = TRUE,
                repetitions = 10,
                project = "new")
```

The `snmf` function estimates an entropy criterion that evaluates the quality of fit of the statistical model to the data using a cross-validation technique (Figure 2). The entropy criterion can help choosing the number of ancestral populations that best explains the genotypic data (Alexander and Lange, 2011; Frichot et al., 2014). Here we have a clear minimum at $K = 4$, suggesting 4 genetic clusters. Often, the plot shows a less clear pattern, and choosing the "knee" is a generally good approach. The number of ancestral populations is closely linked to the number of principal components that explain variation in the genomic data. Both numbers can help determining the number of latent factors when correcting for confounding effects due to population structure in ecological association tests.

```
# plot cross-entropy criterion for all runs in the snmf project
plot(project, col = "blue", pch = 19, cex = 1.2)
```

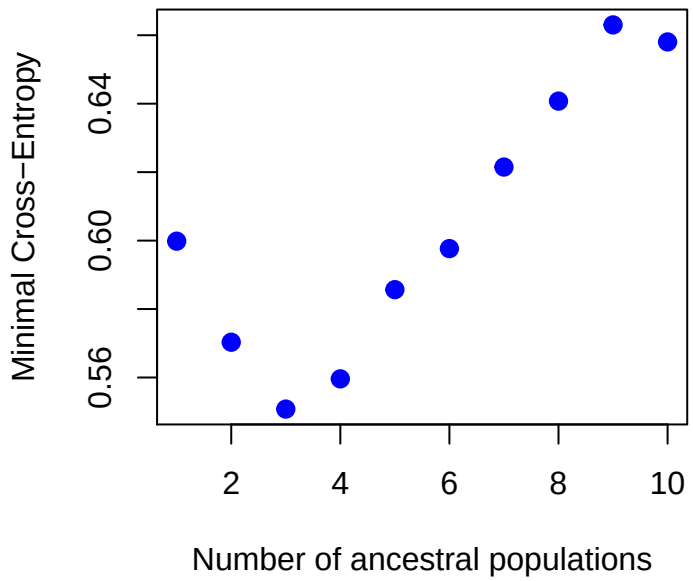

Figure 2: Value of the cross-entropy criterion as a function of the number of populations in `snmf`.

The next step is to display a barplot for the Q-matrix. In Figure 3, the `Q()` function of LEA is called and the output Q-matrix is converted into a `Qmatrix` object. The conversion of the Q-matrix as a `Qmatrix` object is also useful for running improved graphical functions from other packages such as `tess3r` (Caye et al., 2016).

```
# select the best run for K = 4
best = which.min(cross.entropy(project, K = 4))
Q.matrix <- as.qmatrix(Q(project, K = 4, run = best))
my.colors <- c("tomato", "lightblue",
               "olivedrab", "gold")
barplot(Q.matrix, border = NA, space = 0,
        col = my.colors,
        xlab = "Individuals",
        ylab = "Ancestry proportions",
        main = "Ancestry matrix") -> bp
axis(1, at = 1:nrow(Q.matrix),
     labels = bp$order, las=1,
     cex.axis = .3)
```

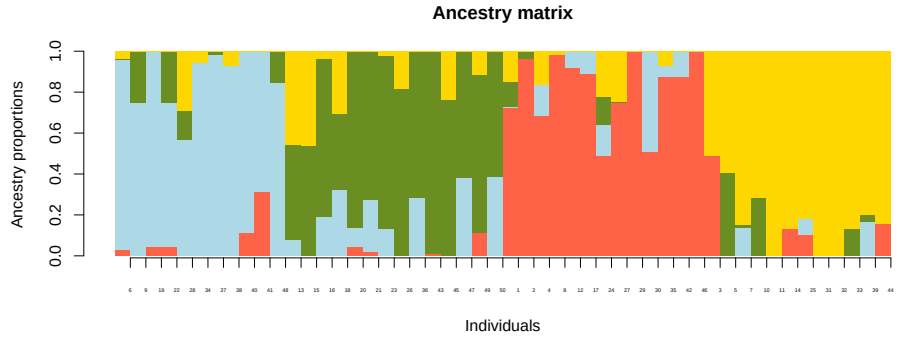

Figure 3: Ancestry coefficients obtained from `snmf`.

3.3 Population differentation tests using `snmf`

The most common approaches to detecting outlier loci from the genomic background have focused on extreme values of the fixation index, F_{st}, across loci. The `snmf()` function can compute fixation indices when the population is genetically continuous, when predefining subpopulations is difficult, and in the presence of admixed individuals in the sample ((Martins et al., 2016)). In the snmf approach, population differentiation statistics are computed from ancestry coefficients obtained a `snmf` project, and p-values are returned for all loci. Figure 4 is an example of outlier analysis with `snmf`.

```
# Population differentiation tests
p = snmf.pvalues(project,
                 entropy = TRUE,
                 ploidy = 2,
                 K = 4)
pvalues = p$pvalues
par(mfrow = c(2,1))
hist(pvalues, col = "orange")
plot(-log10(pvalues), pch = 19, col = "blue", cex = .7)
```

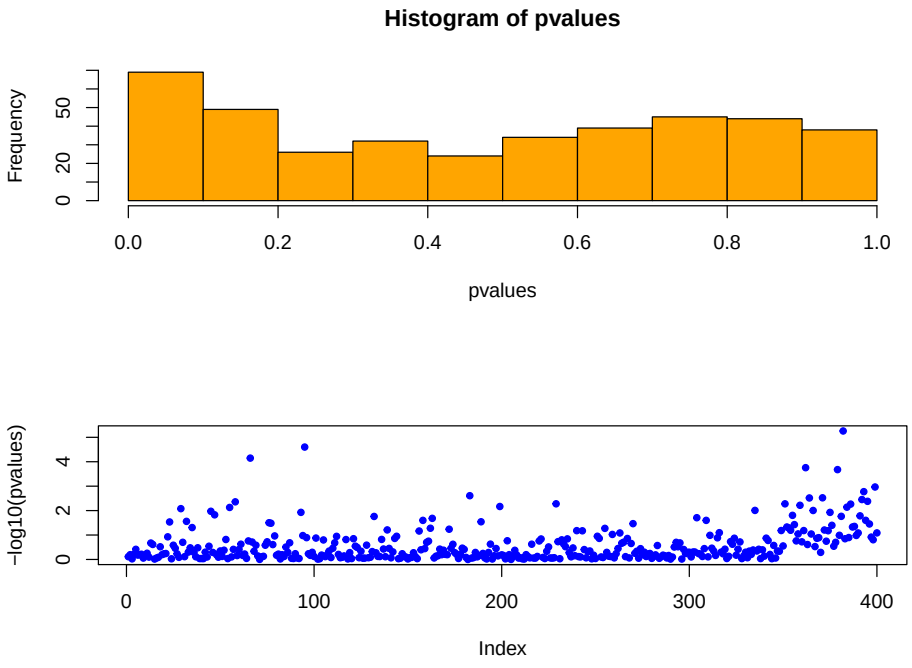

Figure 4: P-values for population differentiation tests.

3.4 Missing genotype imputation using `snmf`

Missing genotypes are critical to genomewide association studies. Before running an association study, an important step is to replace the missing data, represented as '9' in the `geno` and `lfmm`) files, by better values. To provide an example of missing data imputation, let's start by removing 100 genotypes from the original data. The resulting matrix is saved in the file `genotypeM.geno`.

```
# creation of a genotypic matrix  with missing genotypes
dat = as.numeric(tutorial.R)
dat[sample(1:length(dat), 100)] <-  9
dat <- matrix(dat, nrow = 50, ncol = 400)
write.lfmm(dat, "genotypeM.lfmm")

## [1] "genotypeM.lfmm"
```

Next, `snmf` can be run on the data with missing genotypes as follows. The genotypic matrix completion is based on estimated ancestry coefficients and ancestral genotype frequencies.

```
project.missing = snmf("genotypeM.lfmm", K = 4,
        entropy = TRUE, repetitions = 10,
        project = "new")
```

Then the project data can be used to impute the missing data as follows.

```
# select the run with the lowest cross-entropy value
best = which.min(cross.entropy(project.missing, K = 4))

# Impute the missing genotypes
impute(project.missing, "genotypeM.lfmm",
       method = 'mode', K = 4, run = best)

## Missing genotype imputation for K = 4
## Missing genotype imputation for run = 3
## Results are written in the file:  genotypeM.lfmm_imputed.lfmm

# Proportion of correct imputation results
dat.imp = read.lfmm("genotypeM.lfmm_imputed.lfmm")
mean( tutorial.R[dat == 9] == dat.imp[dat == 9] )

## [1] 0.8
```

The results are saved in an output file with the string `"imputed"` in its suffix name.

4 Ecological associations tests using lfmm

The `R` package `LEA` performs genomewide association analysis based on latent factor mixed models using the `lfmm` function (Frichot et al., 2013). To recall the model, let G denote the genotypic matrix, storing allele frequencies for each individual at each locus, and let X denote a set of d ecological predictors or phenotypic traits. LFMMs consider the genotypic matrix entries as response variables in a linear regression model

$$G_{i\ell} = \mu_\ell + \beta_\ell^T X_i + U_i^T V_\ell + \epsilon_{i\ell}\,, \tag{1}$$

where μ_ℓ is a locus specific effect, β_ℓ is a d-dimensional vector of regression coefficients, U_i contains K latent factors, and V_ℓ contains their corresponding loadings (i stands for an individual and ℓ for a locus). The residual terms, $\epsilon_{i\ell}$, are statistically independent Gaussian variables with mean zero and variance σ^2.

In latent factor models, association between predictors and allele frequencies can be tested while estimating unobserved latent factors that model confounding effects. In principle, the latent factors include levels of population structure due to shared demographic history or background genetic variation. After correction for confounding effects, association between allele frequencies and an ecological predictor at a particular locus is often interpreted as a signature of natural selection.

Running LFMM. The `lfmm` program is based on a stochastic algorithm (MCMC) which doesnot provide exact results. We recommend using large number of cycles (e.g., `-i 6000`) and the burnin period should set at least to one-half of the total number of cycles (`-b 3000`). We have noticed that the program results are sensitive to the run-length parameter when data sets have relatively small sizes (eg, a few hundreds of individuals, a few thousands of loci). We recommend increasing the burnin period and the total number of cycles in this situation.

```
# main options:
# K: (the number of latent factors)
# Runs with K = 6 and 5 repetitions.
project = NULL
project = lfmm("genotypes.lfmm",
               "gradients.env",
                K = 6,
                repetitions = 5,
                project = "new")
```

Deciding the number of latent factors. Deciding an appropriate value for the number of latent factors in the `lfmm` call can be based on the analysis of histograms of test significance values. Ideally, histograms should be flat, with a peak close to zero.

Since the objective is to control the false discovery rate (FDR) while keeping reasonable power to reject the null hypothesis, we recommend using several runs for each value of K and combine p-values (use 5 to 10 runs, see our script below). Choosing values of K for which the histograms show their correct shape warrants that the FDR can be controlled efficiently.

Testing all K values in a large range, from 1 to 20 for example, is generally useless. A careful analysis of population structure and estimates of the number of ancestral populations contributing to the genetic data indicates the range of values to be explored. For example the `snmf` program estimates 4 ancestral populations, then running `lfmm` for $K = 3 - 6$ often provides good results.

Combining z-scores obtained from multiple runs. We use the Fisher-Stouffer method to combine z-scores from multiple runs. In practice, we found that using the median z-scores of 5-10 runs and re-adjusting the p-values afterwards can increase the power of `lfmm` tests. This procedure is implemented in `LEA` function `lfmm.pvalues()`.

```
# compute adjusted p-values
p = lfmm.pvalues(project, K = 6)
pvalues = p$pvalues
```

The results displayed in Figure 5 show that the null-hypothesis is correctly calibrated. The loci exhibiting significant associations are found at the right on the Manhattan plot.

```
# GWAS significance test
par(mfrow = c(2,1))
hist(pvalues, col = "lightblue")
plot(-log10(pvalues), pch = 19, col = "blue", cex = .7)
```

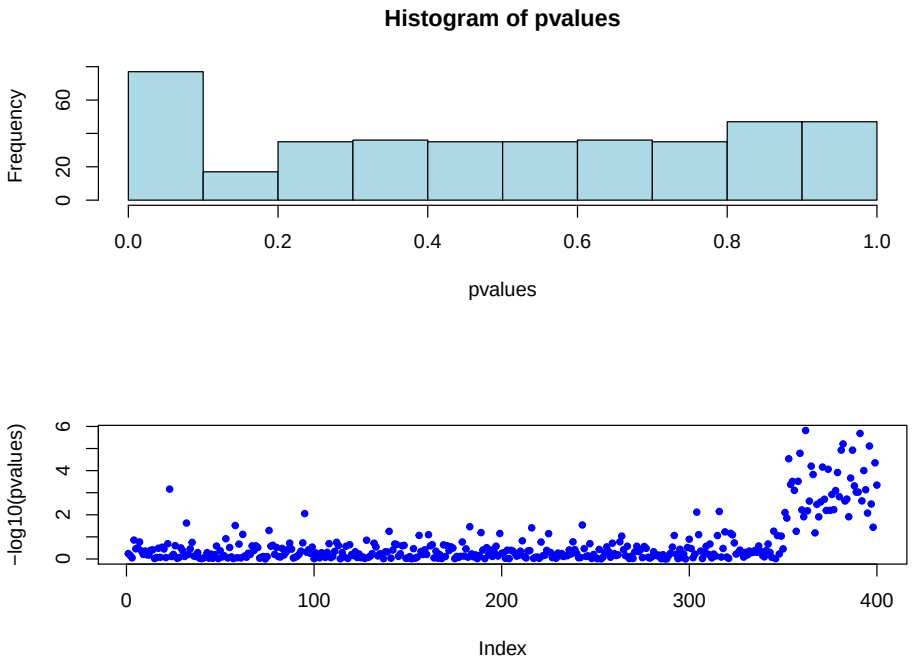

Figure 5: P-values for LFMM tests. The loci showing significant associations are at the right on the Manhattan plot.

To adjust p-values for multiple testing issues, we use the Benjamini-Hochberg procedure (Benjamini and Hochberg, 1995). We set the expected levels of FDR to $q = 5\%$, 10%, 15% and 20% respectively . The lists of candidate loci are given by the following script. Since we the ground truth is known for the simulated data, we can compare the expected FDR levels to their observed levels, and compute the power (TPR) of the test.

```
for (alpha in c(.05,.1,.15,.2)) {
    # expected FDR
    print(paste("Expected FDR:", alpha))
    L = length(pvalues)

    # return a list of candidates with expected FDR alpha.
    # Benjamini-Hochberg's algorithm:
```

```
    w = which(sort(pvalues) < alpha * (1:L) / L)
    candidates = order(pvalues)[w]

    # estimated FDR and True Positive Rate
    Lc = length(candidates)
    estimated.FDR = sum(candidates <= 350)/Lc
    print(paste("Observed FDR:",
                round(estimated.FDR, digits = 2)))
    estimated.TPR = sum(candidates > 350)/50
    print(paste("Estimated TPR:",
                round(estimated.TPR, digits = 2)))
}
```

```
## [1] "Expected FDR: 0.05"
## [1] "Observed FDR: 0.03"
## [1] "Estimated TPR: 0.72"
## [1] "Expected FDR: 0.1"
## [1] "Observed FDR: 0.08"
## [1] "Estimated TPR: 0.88"
## [1] "Expected FDR: 0.15"
## [1] "Observed FDR: 0.08"
## [1] "Estimated TPR: 0.94"
## [1] "Expected FDR: 0.2"
## [1] "Observed FDR: 0.1"
## [1] "Estimated TPR: 0.94"
```

References

Alexander DH and Lange K. 2011. Enhancements to the ADMIXTURE algorithm for individual ancestry estimation. BMC Bioinformatics 12:246.

Benjamini Y and Hochberg Y. 1995. Controlling the false discovery rate: a practical and powerful approach to multiple testing. J R Stat Soc B Met. pp. 289–300.

Caye K, Jay F, Michel O and Francois O. 2016. Fast inference of individual admixture coefficients using geographic data. bioRxiv p. 080291.

François O and Durand E. 2010. Spatially explicit bayesian clustering models in population genetics. Mol Ecol Resour. 10:773–784.

Frichot E and Francois O. 2015. LEA: an R package for Landscape and Ecological Association studies. Methods in Ecology and Evolution 6:925–929.

Frichot E, Mathieu F, Trouillon T, Bouchard G and François O. 2014. Fast and efficient estimation of individual ancestry coefficients. Genetics 196:973–983.

Frichot E, Schoville SD, Bouchard G and François O. 2013. Testing for associations between loci and environmental gradients using latent factor mixed models. Mol Biol Evol. 30:1687–1699.

Martins H, Caye K, Luu K, Blum MGB and Francois O. 2016. Identifying outlier loci in admixed and in continuous populations using ancestral population differentiation statistics. Molecular Ecology 25:5029–5042.

Patterson N, Price AL and Reich D 2006. Population structure and eigenanalysis. PLoS Genet. 2:20.

Pritchard JK, Stephens M and Donnelly P. 2000. Inference of population structure using multilocus genotype data. Genetics 155:945–959.