LEA: An R Package for Landscape and Ecological Association Studies

Eric Frichot and Olivier François
Université Joseph Fourier Grenoble 1,
Centre National de la Recherche Scientifique,
TIMC-IMAG UMR 5525, Grenoble, 38042, France.

Contents

1 Overview

LEA (?) is an R package dedicated to landscape genomics and ecological association tests. LEA can run analyses of population structure and genome scans for local adaptation. It includes statistical methods for estimating ancestry coefficients from large genotypic matrices and evaluating the number of ancestral populations (snmf, pca); and identifying genetic polymorphisms that exhibit high correlation with some environmental gradient or with the variables used as proxies for ecological pressures (lfmm), and controlling the false discovery rate. LEA is mainly based on optimized C programs that can scale with the dimension of very large data sets.

2 Introduction

The goal of this tutorial is to give an overview of LEA functionalities.

One specifity of the LEA package is to be able to handle very large population genetic data sets. Genomic data are never loaded into the R memory, and they are processed using fast C codes wrapped into the R code. For this reason, most of the LEA functions use character strings containing paths to input files as arguments.

As some functions may take several hours for very large datasets, nobody wants to erase their results when quitting R. That is why output files are written into text files that can be read by LEA after each run. We advise creating a working directory containing your data when you start using LEA. It is assumed that two files with the same name and a different extension in the same directory contain the same data matrix in different formats.

```
> # creation of a directory for the LEA analyses
> dir.create("LEA_analyses")
> # set this new directory as the working directory
> setwd("LEA_analyses")
```

2.1 Preparing the data

The `R` package `LEA` can handle several classical formats for input files of genotypic matrices. More specifically, the package uses the `lfmm` and `geno` formats, and provides functions to convert from `ped`, `vcf`, and `ancestrymap` formats. While the `lfmm` and `geno` formats usually encode SNP data, those formats can also be used for coding amplification fragment length polymorphisms and microsatellite markers. In addition to genotypic matrices, `LEA` can also process allele frequency data when they are encoded in the `lfmm` formats. Ecological variables must be formatted in the `env` format used by the computer program `lfmm` (**?**).

The `LEA` package can handle missing data, but the algorithm used for genotype imputation is basic. We encourage users having more than 10 missing genotypes in their data to input the missing data issue by using matrix completion or genotype imputation programs such as IMPUTE2 or MENDEL-IMPUTE before starting their analyses with `LEA`. Ecological data must be prepared using the env format. 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 (or similar approaches) and using the first components as new ecological variables is one of these approaches.

The tutorial data set is composed of a genotypic matrix called tutorial.R with 50 individuals for 400 SNPs. The last 50 SNPs are correlated with an environmental variable called tutorial.C. This dataset is a subset of the dataset displayed in the note associated with the package (**?**).

```
> library(LEA)
> # Creation of the genotypic file "genotypes.lfmm"
> # with 400 SNPs for 50 individuals.
> data("tutorial")
> # in the lfmm format
> write.lfmm(tutorial.R, "genotypes.lfmm")
> # in the geno format
> write.geno(tutorial.R, "genotypes.geno")
> # creation of the environment file, gradient.env.
> # It contains 1 environmental variable for 50 individuals.
```

```
> write.env(tutorial.C, "gradients.env")
```

3 Analysis of population structure

The R package LEA implements two classical approaches for the estimation of population genetic structure: principal component analysis (pca) and admixture analysis (**??**). The algorithms programmed in LEA are improved versions of pca and admixture analysis able to process very 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 scree-plot 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 calculated),
> #                    center and scale.
> # Creation of   genotypes.pcaProject - the pcaProject object.
> #               a directory genotypes.pca containing:
> # 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 scree-plot (Figure 1) or computing Tracy-Widom tests with the LEA function tracy.widom (**?**).

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

> # display the p-values for the Tracy-Widom tests.
> tw$pvalues[1:10]

 [1] 8.000e-09 8.000e-09 8.000e-09 1.503e-04 3.152e-02 4.215e-01 6.565e-01
 [8] 6.859e-01 6.738e-01 9.363e-01
```

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

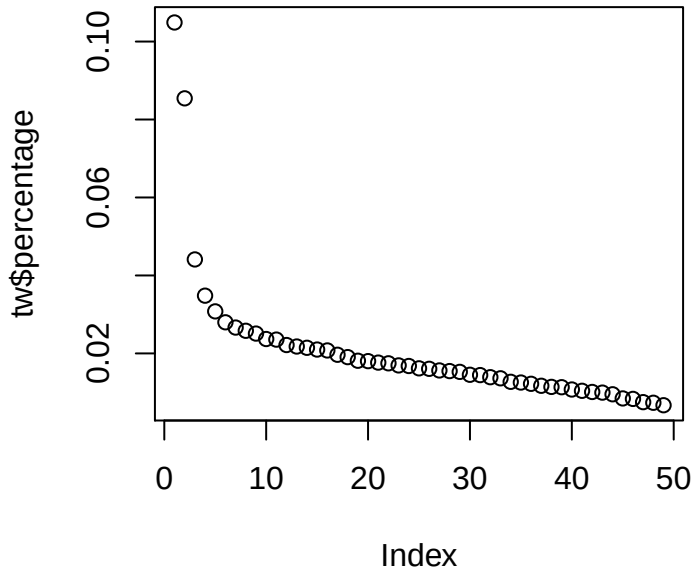

Figure 1: plot of the percentage of variance explained by each component

3.2 Inference of individual admixture coefficients using `snmf`

Similar to Bayesian clustering programs, `LEA` includes an `R` function to estimate individual admixture coefficients from the genotypic matrix (**??**). Assuming K ancestral populations, the `R` function `snmf` provides least-squares estimates of ancestry proportions (**?**).

```
> # main options, K: (the number of ancestral populations),
> #        entropy: calculate the cross-entropy criterion,
> #        CPU: the number of CPUs.
>
> # Runs with K between 1 and 10 with cross-entropy and 10 repetitions.
> project = NULL
> project = snmf("genotypes.geno", K=1:10, entropy = TRUE, repetitions = 10,
+     project = "new")
```

The `snmf` function also 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 (**??**).

```
> # plot cross-entropy criterion of all runs of the project
> plot(project, lwd = 5, col = "red", pch=1)
```

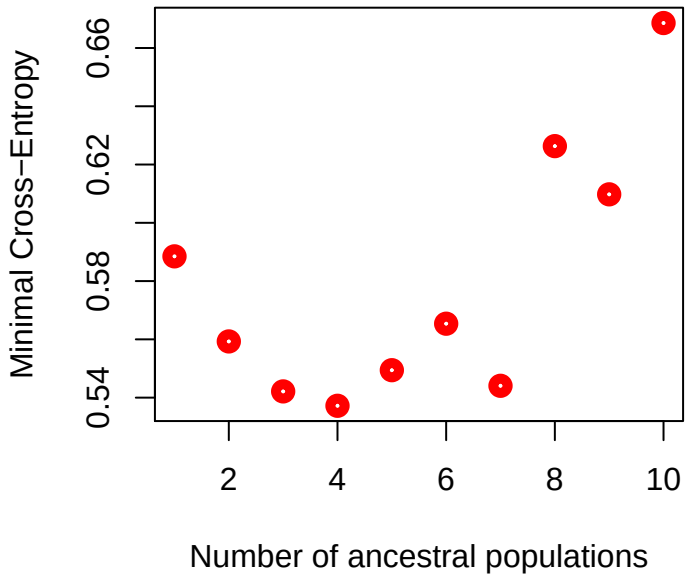

Figure 2: Value of the cross-entropy criterion as a function of the number of factors in `snmf`

```
> # get the cross-entropy of each run for K = 4
> ce = cross.entropy(project, K = 4)
> # select the run with the lowest cross-entropy
> best = which.min(ce)
```

The number of ancestral population 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.

4 Ecological associations tests using lfmm

The `R` package `LEA` performs ecological association tests based on latent factor mixed models (`LFMM`, `?`). Let G denote the genotypic matrix, storing allele frequencies for each individual at each locus, and let X denote a set of d ecological variables. LFMMs consider 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, associations between ecological variables 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, significant association between allele frequencies and an observed ecological variable is often interpreted as evidence for selection at a particular locus.

The run. The `lfmm` program is based on a stochastic algorithm (MCMC) which cannot provide exact results. We recommend using large number of cycles (e.g., `-i 10000`) and the burn-in period should set at least to one-half of the total number of cycles (`-b 5000`). 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 burn-in period and the total number of cycles in this situation.

```
> # main options, K: (the number of latent factors),
> #           CPU: the number of CPUs.
>
> # Runs with K = 6 and 5 repetitions.
> # The runs are composed of 6000 iterations including 3000 iterations
> # for burnin.
> # around 20 seconds per run.
> 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 `lfmm` can be based on the analysis of histograms of test p-values. Here, the objective is to control the false discovery

rate while keeping reasonable power to reject the null hypothesis. To choose the number of factors, we suggest using the genomic inflation factor. According to Devlin and Roeder (1999), this quantity is defined as

$$\lambda = \text{median}(z^2)/0.456\,.$$

The inflation factor usually decreases with increasing values of K. To compute the genomic inflation factor, we recommend using several runs for each value of K and taking the median or the mean of the λ values obtained from the above formula (use 5 to 10 runs, see our script below). Choosing values of K for which the estimate of λ is close to (or slightly below) 1.0 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. If for example the `snmf` or `ADMIXTURE` programs estimate 5 ancestral populations, then running `lfmm` $K = 4 - 7$ often provides inflation factors close to 1.0.

Combining z-scores obtained from multiple runs. We suggest using the Fisher-Stouffer or a similar 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 increase the power of `lfmm` tests. This can be done by using the `LEA` function, `adjusted.pvalues`.

p-values were adjusted as follows

```
> # calculate adjusted p-values
> res = adjusted.pvalues(project, K = 6)
> cp.values = res$p.values
> gif = res$genomic.inflation.factor
```

To adjust p-values for multiple testing issues, we used the Benjamini-Hochberg procedure with expected levels of FDR equal to $q = 5\%$, 10%, 15% and 20% respectively (**?**). The lists of candidate loci is given by

```
> for (alpha in c(.05,.1,.15,.2)) {
+     # expected FDR
+     print(paste("expected FDR:", alpha))
+     L = length(cp.values)
+     # return a list of candidates with an expected FDR of alpha.
+     w = which(sort(cp.values) < alpha * (1:L) / L)
+     candidates = order(cp.values)[w]
+
+     # estimated FDR and True Positif
```

```
+     estimated.FDR = length(which(candidates <= 350))/length(candidates)
+     estimated.TP = length(which(candidates > 350))/50
+     print(paste("FDR:", estimated.FDR, "True Positive:", estimated.TP))
+ }
```

```
[1] "expected FDR: 0.05"
[1] "FDR: 0.0263157894736842 True Positive: 0.74"
[1] "expected FDR: 0.1"
[1] "FDR: 0.0238095238095238 True Positive: 0.82"
[1] "expected FDR: 0.15"
[1] "FDR: 0.0416666666666667 True Positive: 0.92"
[1] "expected FDR: 0.2"
[1] "FDR: 0.0408163265306122 True Positive: 0.94"
```

5 Session info

Here is the output of sessionInfo on the system on which this document was compiled:

- R version 3.1.2 (2014-10-31), `x86_64-pc-linux-gnu`
- Locale: `LC_CTYPE=en_US.UTF-8`, `LC_NUMERIC=C`, `LC_TIME=en_US.UTF-8`, `LC_COLLATE=en_US.UTF-8`, `LC_MONETARY=en_US.UTF-8`, `LC_MESSAGES=en_US.UTF-8`, `LC_PAPER=en_US.UTF-8`, `LC_NAME=C`, `LC_ADDRESS=C`, `LC_TELEPHONE=C`, `LC_MEASUREMENT=en_US.UTF-8`, `LC_IDENTIFICATION=C`
- Base packages: base, datasets, graphics, grDevices, methods, stats, utils
- Loaded via a namespace (and not attached): tools 3.1.2