Inferring patterns of migration from gene frequencies with R (or S-Plus), Version 1.2.0

Jarle Tufto

August 28, 2001

Abstract

A collection of R (and S-Plus) functions and some additional c-code for estimating the pattern of migration in a subdivided population from genetic differences generated by local genetic drift is described. The method is applicable to migration matrices of any form. Submodels defining the general form of the migration matrix can be written by the user. Functions are also provided for carrying out likelihood ratio tests between alternative models such as the island model and the stepping stone model, or between alternative user defined submodels. In addition, plots of predicted and observed covariances can be made to assess the fit of a selected model. Finally, code for computing the asymptotic eigenvalue variance effective size of a subdivided population is provided.

Contents

1 Introduction

In population genetics, there has long been an interest in understanding and analyzing the gene frequency structure found in natural populations. Local genetic drift will tend to create genetic differences between different subunits of a population. These differences will, after a number of initial generations, be balanced by migration. One can thus, in principle, estimate the amount of migration between the different subunits from the observed gene frequencies. Most earlier approaches to this problem have, in part, relied on the assumptions of the island model. Under that model one can estimate the number of migrants Nm from the estimated amount of genetic differentiation as measured by e.g. Wright's parameter F_{st}. Typically, these estimates are then, in further exploratory analysis, regressed against geographic distances to test for isolation by distance effects predicted by some idealized theoretical models. This manual documents the use of some S-Plus and c code I wrote to implement an alternative solution of this problem based more directly on standard statistical principles, which allows much less restrictive assumptions about the underlying pattern of migration. The general method, which is based on an idea in Felsenstein (1982), is described in two papers (Tufto et al., 1996, 1998; Tufto and Hindar, 2002); preprints of these are available at `http://www.math.ntnu.no/~jarlet/`. A brief summary of the approach is also given here.

You can read the online, most up-to-date version of this manual at `http://www.math.ntnu.no/~jarlet/migration`. It is assumed that the reader has access to and some familiarity with R or S-plus.

2 The model

We consider $i = 1, 2, \dots, n$ subpopulation. The available data from which the inferences are to be drawn are vectors of gene (allele) frequencies in these subpopulations, at a number of different loci. Differences between local gene frequencies is assumed to be a result of local genetic drift balanced by migration (or mutation). The pattern of migration between these subpopulation is described by a migration matrix $\mathbf{M}$ with elements m_{ij} denoting the probability, or the proportion, of genes which originates from subpopulation j, given migration to subpopulation i. The objective of analysis is to obtain an estimate of the migration matrix $\mathbf{M}$. In general, we will be interested in migration matrices of certain biologically interesting forms only, involving a small number of parameters, which we hope to be able to estimate. The effective size, or at least, the relative size, of each subpopulation, N_i, is treated as known parameters. All subpopulations are also assumed to receive a small amount of immigrant genes from a large 'outside world' population with gene frequency q.

With these assumptions, one can show that the gene frequencies, at equilibrium, provided that the fluctuations around q are small, have covariance matrix **C**, satisfying the matrix equation

$$\mathbf{C} = \mathbf{M}\mathbf{C}\mathbf{M}^T + \mathbf{E}, \tag{1}$$

where

$$e_{ij} = \begin{cases} \frac{1}{2N_i} & \text{for } i = j \\ 0 & \text{for } i \neq j, \end{cases} \tag{2}$$

Using the solution of (1), the idea is to compute the likelihood of the data approximately by assuming multivariate normality. It is then possible to obtain maximum likelihood estimates of the parameters of any model for the migration matrix using numerical methods.

3 Downloading and loading the code into R or S-Plus

The library comes in one version which detects whether it is running under R or S-plus. In theory, everything should work in both environments. However, because the S-plus' memory usage is sometimes quite inefficient (see section 5 and 8), and because S-Plus is pretty expensive, proprietary software whereas R is GPL'ed software freely available on the Internet (e.g. from `http://www.R-project.org`), the code optimized for R. Note that to use the R version, you will probably need to have access to R installed on a Unix system which supports loading of shared libraries with the "dlopen" mechanism. A good idea is probably to install some popular Linux distribution on a fast computer somewhere in your network neighbourhood, install R, and run everything remotely over the network. If you want to use R's excellent graphics capabilities you will also want an X-server on your computer instead of just telnet'ing to the remote computer. There are several freeware and shareware version available on the net (e. g at `http://www.starnet.com` and `http://www.frontiertech.com`).

The R (or S-plus) and c code needed consists of the two files; `migrlib.R` and `migrlib.c`. In addition, the c part of the code uses `randlib` for random number generation. These files and accompanying documentation (this file in postscript format) can be downloaded from `http://www.math.ntnu.no/~jarlet/migration/migrlib.tar.gz` (see Table 1).

Table 1: Contents of `migrlib.tar.gz`

`migrlib.R`	R (or S-Plus) functions
`migrlib.c`	c-code for generating bootstrap samples
`migrlib-doc.tex`	LaTeX file for generating this documentation
`migrlib-doc.pdf`	This documentation in portable document format
`randlib.c`	c-library for random number generation
`randlib.h`	
`linpack.c`	
`com.c`	

The `randlib` library may also be downloaded separately from from `http://hpftp.cict.fr/hppd/hpux/Maths/Misc/randlib-1.3/`.

The first thing to do is to compile the c routines together with `randlib`. On the Unix system I used at one point in time, which had S-plus installed, there was a special command for compiling c code for later loading into S-plus. The following command

```
s+compile -o all.o -g linpack.c com.c randlib.c migrlib.c
```

from the OS-prompt, seemed to do the trick, but this will vary between different operating systems. This will probably be different on, for example, a Windows 95 platform. It should be described somewhere in S-plus' documentation.

Dynamic loading of shared libraries in R is presently only implemented on unix version, for example linux. On this platform, the shared library "all.so" can be compiled as follows:

```
gcc -g -O2 -fpic -c *.c
ld -shared -o all.so *.o
```

After having compiled the c code, start S-plus (or R), and load the library into R (or into S-Plus) with the command

```
> source("migrlib.R")
```

In doing this, the compiled c code is also loaded through the command `dyn.load.shared("./all.o")` (or `dyn.load("./all.so")` in R).

4 An introductory session

Instead of analysing a real set of data, we will, to illustrates some of the possibilities of the library, first generate a simulated set of data from a known model. To do this let us first create a migration matrix using the `steppingstone` function. This function returns a stepping stone like $(n \times n)$ migration matrix of the following form

$$\mathbf{M} = (1-u)\begin{bmatrix} 1-\frac{1}{2}m_0 & \frac{1}{2}m_0 & 0 & \\ \frac{1}{2}m_0 & 1-m_0 & \frac{1}{2}m_0 & 0 \\ \ddots & \ddots & \ddots & \ddots \\ & 0 & \frac{1}{2}m_0 & 1-\frac{1}{2}m_0 \end{bmatrix}, \tag{3}$$

which is what was used in Tufto et al. (1996). It takes two arguments; the first is the parameter vector `theta` (with elements representing u and m_0), and the second is `n`, the number of subpopulations. Thus,

```
> M <- steppingstone(c(.1,.2),10)
```

generates a migration matrix with $u = .1$ and $m_0 = .2$, with dimension (10×10), that is,

```
> M
      [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10]
 [1,] 0.81 0.09 0.00 0.00 0.00 0.00 0.00 0.00 0.00  0.00
 [2,] 0.09 0.72 0.09 0.00 0.00 0.00 0.00 0.00 0.00  0.00
 [3,] 0.00 0.09 0.72 0.09 0.00 0.00 0.00 0.00 0.00  0.00
 [4,] 0.00 0.00 0.09 0.72 0.09 0.00 0.00 0.00 0.00  0.00
 [5,] 0.00 0.00 0.00 0.09 0.72 0.09 0.00 0.00 0.00  0.00
 [6,] 0.00 0.00 0.00 0.00 0.09 0.72 0.09 0.00 0.00  0.00
 [7,] 0.00 0.00 0.00 0.00 0.00 0.09 0.72 0.09 0.00  0.00
 [8,] 0.00 0.00 0.00 0.00 0.00 0.00 0.09 0.72 0.09  0.00
 [9,] 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.09 0.72  0.09
[10,] 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.09  0.81
```

In addition, let's assume that each subpopulation has effective size $N_e = 100$ individuals. This is done by creating a vector `Ne` with elements specifying the effective size of each subpopulation:

```
> Ne <- rep(100,10)
> Ne
 [1] 100 100 100 100 100 100 100 100 100 100
```

We can now create some data by stochastic simulation using the `simulate` function. Required arguments are a matrix specifying the migration rates and the vector specifying effective populations sizes. If no additional optional arguments are given, a single vector of gene frequencies is returned:

```
> simulate(M,Ne)
      [,1]  [,2] [,3] [,4]  [,5]  [,6]  [,7] [,8]  [,9] [,10]
[1,] 0.285 0.315 0.44 0.61 0.655 0.495 0.495 0.46 0.575 0.425
```

`M` and `Ne` must, of course, have matching dimensions. To simulate data from more than one locus, the optional argument `nloci` should be given. A matrix of gene frequencies is then returned, where each row represent different loci. In this example, the returned matrix is stored in `p`:

```
> p <- simulate(M,Ne,nloci=10)
> p
       [,1]  [,2]  [,3]  [,4]  [,5]  [,6]  [,7]  [,8]  [,9] [,10]
 [1,] 0.520 0.455 0.505 0.640 0.730 0.715 0.455 0.560 0.630 0.515
 [2,] 0.635 0.535 0.485 0.530 0.610 0.450 0.370 0.415 0.515 0.510
 [3,] 0.535 0.375 0.295 0.355 0.475 0.570 0.715 0.610 0.665 0.660
 [4,] 0.615 0.555 0.535 0.550 0.485 0.565 0.470 0.475 0.325 0.465
 [5,] 0.390 0.440 0.445 0.435 0.550 0.510 0.385 0.385 0.610 0.650
 [6,] 0.705 0.580 0.490 0.410 0.565 0.480 0.565 0.745 0.750 0.755
 [7,] 0.445 0.365 0.340 0.475 0.575 0.670 0.640 0.635 0.580 0.585
 [8,] 0.490 0.395 0.545 0.395 0.310 0.320 0.390 0.325 0.280 0.250
 [9,] 0.705 0.625 0.570 0.610 0.410 0.460 0.395 0.335 0.245 0.145
[10,] 0.320 0.300 0.400 0.255 0.320 0.450 0.440 0.425 0.495 0.295
```

In real life, one would typically load the gene frequency data matrix into R (or S-Plus) by doing something like:

```
> p <- matrix(scan("data.txt"),byrow=T,nrow=10)
```

Using the approximation of the likelihood given in the appendix of Tufto et al. (1998), the maximum likelihood estimates of the parameters, for our simulated data set, can be found using the `fitmodel` function. This function takes two arguments; the name `FUN` of the migration model, and the gene frequency matrix:

```
> fit1 <- fitmodel(steppingstone,p)
```

(The above call consumes about 33 seconds of cpu-time on the SGI Indy unix box I have been using. With R running under Linux on a 90MHz Pentium processor it consumes about 25 seconds.)

The `$parameter` component of the returned list contains the maximum likelihood estimates:

```
> fit1$par
[1] 0.004024386 0.128945698
```

and the `$objective` component contains the maximum likelihood:

```
> fit1$obj
[1] 125.4102
```

As illustrated by this example, the estimates obtained are, as they should, not excessivly far from the true values. If you sit down in front of your computer and actually do the above, you will of course get slightly different answers, which should come as any surprise, given that the data is random.

The `fitmodel` function is only a simple interface to, depending on whether you use R or S-plus, S-Plus' non-linear optimizer `nlminb` or R's `nlm` function. The list returned by `nlminb` or `nlm` is in both cases slightly modified so that the `$objective` component contains the maximum log likelihood and the `parameter` component contains the maximum likelihood estimates.

5 Bootstrapping

The `bootstrap` function makes use of both the `simulate` and `fitmodel` functions and performs parametric bootstrapping from a given point in the parameter space. It takes four arguments; the name of the migration model, a parameter vector specifying at which point in the parameter space to simulate from, the observed gene frequency matrix, and, optionally, the number of bootstrap replicates to produce (which by default is 100).

The distribution of the data, conditioned on the observed gene frequency mean at each loci, is simulated. This is why the observed gene frequency matrix is needed by `bootstrap` (to compute the means). 50 bootstrap replicates from the maximum likelihood estimates obtained in section 4 is produced by the following call: [1]

```
> boots <- bootstrap(steppingstone,fit1$par,p,nboots=50)
```

If you get the feeling that this call has caused your computer to hang, you can set the optional argument `dots=T`. Then, at least, some small dots start to appear every now and then.

The `bootstrap` function returns a matrix with rows corresponding to each bootstrap replicate of the parameter vector:

```
> boots
              [,1]       [,2]
 [1,] 0.002350949 0.12999907
 [2,] 0.002001756 0.09736528
 [3,] 0.001058648 0.14776179
 [4,] 0.009843646 0.11854703
   .        .          .
   .        .          .
```

The variance-covariance matrix of the replicates is thus easy to compute using S-Plus' function

---

[1] If you're running the S-plus version, you should also note that this command may cause S-plus to consume huge amounts of memory — after hours of simulation S-plus may suddenly decide to terminate because of too little memory. Note that these problems with S-plus also turns up in likelihood ratio tests (see section 8). One solution to this is to call `bootstrap` several times with `nboots` set to some small number (e.g. 50 if you have about 64MB of memory). You may also try to run the the R-version which does not have these problems with memory.

```
> var(boots)
              [,1]          [,2]
[1,]  5.317043e-05 -0.0000885599
[2,] -8.855990e-05  0.0007198054
```

Taking the square root of the diagonal elements gives us an estimate of the standard errors

```
> sqrt(diag(var(boots)))
[1] 0.007291805 0.026829190
```

which aren't to large — the coefficient of variation for m_0 is only 0.2. The uncertainty in u is quite big though.

6 Bias correction

Bias is easily estimated as the difference between the original estimates and the mean of the bootstrap replicates:

```
> bias <- apply(boots,2,mean) - fit1$par
> bias
[1] 0.001642839 0.006198921
```

If the bias not excessivly large, and if it appears to be approximately independent of the true parameter values, one way to correct for bias (Efron and Tibshirani, 1993, p. 138) is to subtract the estimated bias from the orginal estimates:

```
> fit1$par - bias
[1] 0.002381547 0.122746777
```

In many cases, however, especially if there is a large amount of between-population differentiation, there may be considerable bias in the obtained estimate, not only because asymptotic theory doesn't apply, but also because the likelihood function is quite approximate. More sophisticated methods for reducing bias such as that of Cabrera and Watson (1997) may then be needed. I will try to tidy up the code I used in Tufto et al. (1998) and put it into the library. The theoretical justification for that method, however, only apply to one-parametric models.

7 Writing your own migration model

7.1 General ideas

The main motivation for developing the approach has been to make it possible for different researchers to analyze their data using a model that they find biologically interesting and realistic. For example, depending on the mode of dispersal, the probability distribution of the gene displacements may decay more or less exponentially with distance. There may also be effects of habitat selection; migration to high quality subpopulation may be more likely, and the geography may more complicated than in idealized theoretical models.

Parameters representing such dependencies are fully possible to incorporate in the analysis. The idea is that different users should write their own function for a migration matrix model that they find biologically interesting. This function should take the following two arguments; a parameter vector `theta`, and `n`, the number of subpopulations, and should return a $(\mathtt{n} \times \mathtt{n})$ matrix. The sum of each row must be equal to or less than one. The remaining proportion, that is, $1 - \sum_{j=1}^{n} m_{ij}$, represent the sum the mutation rate and the rate of migration from the 'outside world' into each subpopulation i.

Both the R and S-plus version provides a way of handling so called box constraints on the parameters. For example, in the case of the `steppingstone` model, only parameter values in the range from 0 to 1 are allowed. When the likelihood is maximised, `fitmodel` needs to get this information from the function defining the migration model provided by the user. The idea is to define this information once and for all inside the declarations for the new migration model. The mechanism for achieving this is implemented somewhat differently in the R and S-plus version:

7.2 R version

The R version uses R's `nlm` function to maximise the likelihood numerically. `nlm` does not take box constaints. The idea is therefore to work with parameters transformed onto the whole real line, for example by logit transformation of probabilities (or proportions) (such as u and m_0 of the steppingstone model). The user defined function should therefore be able to handle transformed paremeter when the additional argument `trans=T`. In addition, when the other additional argument `convert=T`, the user defined function defining the migration pattern should return, instead of a migration matrix, the parameter vector converted back to either the untransformed or transformed state, depending on whether `trans=T` or `trans=F`, respectively. Also, when the optional argument `start=T`, a vector of sensible (untransformed) starting parameter values should be returned. You are free to implement this in any way you like inside your own function. The R version of `steppingstone` should give you an idea of how to do it:

```
steppingstone <- function(theta,n,trans=F,start=F,convert=F)
{
   if (trans) {theta <- c(ilogit(theta[1],lower=.0001),ilogit(theta[2]))}
   if (convert) {
    if (!trans) {theta <- c(logit(theta[1],lower=.0001),logit(theta[2]))}
    return(theta)
   }
   if (start) return(c(.1,.1))

   M <- matrix(0,ncol=n,nrow=n)
   diag(M) <- 1 - theta[2]
   M[abs(row(M)-col(M))==1] <- theta[2]/2
   M[1,1] <- M[n,n]  <- 1 - theta[2]/2
   M <- (1-theta[1])*M
   return(M)
}
```

Logit and log transformations and their inverses are provided by the functions `logit`, `ilogit`, `log`, and `ilog` which are part of `migrlib.r`. In some cases is necessary to restict the lower bound of the parameter range to some small postive value, e.g. 0.0001 instead of 0 (see Tufto et al., 1998, p. 1979 for details). This lower bound can be passed to `logit`, `ilogit`, `log`, and `ilog` through the optional `lower` argument.

7.3 S-plus version

`fitmodel` does, as already mentioned, use `nlminb` to carry out the actual numerical maximization. In addition to the name of the function that is to be optimized, `nlminb` also needs vectors specifying starting parameter values and upper and lower box constraints. The `fitmodel` function assumes that these vectors will be returned through additional calls to `FUN` with one of the optional optional argument `start`, `lower`, or `upper` set true. The following call illustrates the correct behaviour of `steppingstone`:

```
> steppingstone(upper=T)
[1] 1 1
```

The source code to achieve this in the case of the `steppingstone` model is as follows:

```
steppingstone <- function(theta,n,upper=F,lower=F,start=F)
{
   if (upper) {return(c(1.0,   1.0 )) }
   if (lower) {return(c(0.0001,0.0 )) }
   if (start) {return(c(0.1,   0.1 )) }

   M <- matrix(0,ncol=n,nrow=n)
   diag(M) <- 1 - theta[2]
   M[abs(row(M)-col(M))==1] <- theta[2]/2
   M[1,1] <- M[n,n]  <- 1 - theta[2]/2
   M <- (1-theta[1])*M
   return(M)
}
```

Only when none of the optional arguments are true (which is default) is the main code executed and a migration matrix returned. Although this perhaps isn't very good programming practice, the main advantage of implementing things this way is that good starting and box constraint vectors are defined once and for all, and that functions carrying out, for example, bootstrapping and likelihood ratio tests can retrieve this information only knowing the name of the migration matrix submodel. This will prove very useful when working with several alternative models.

7.4 Some additional remarks

Note that user provided functions should return a *backward* migration matrix. You can, however, in cases where this is more natural, specify a model for the forward matrix, and then, before returning the resulting matrix, convert it to a backward matrix, with the `forw2back` function. This function takes a forward matrix as its first argument and a vector of subpopulation sizes as its second argument and returns a backward matrix (Tufto et al., 1998, eq. 5).

User provided functions specifying a model for the migration matrix may, of course, incorporate dependencies on additional information, say the geographic location (e.g. Tufto et al., 1998) or habitat quality of different subpopulations, through the use of global variables.

8 Likelihood ratio tests

Code is also provided for doing likelihood ratio tests between different nested, and perhaps also, non-nested alternative models. This is done, by brute force, by function `lrtest` by simulating bootstrap data from H_0 and computing the likelihood ratio by fitting both H_0 and H_1 numerically to each bootstrap data set.

Let's say we want to use the island model as our null hypothesis. Function `islandmodel` simply returns a diagonal matrix with nonzero elements equal to $1-m$:

```
> islandmodel(.1,10)
      [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10]
 [1,]  0.9  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0   0.0
 [2,]  0.0  0.9  0.0  0.0  0.0  0.0  0.0  0.0  0.0   0.0
 [3,]  0.0  0.0  0.9  0.0  0.0  0.0  0.0  0.0  0.0   0.0
 [4,]  0.0  0.0  0.0  0.9  0.0  0.0  0.0  0.0  0.0   0.0
 [5,]  0.0  0.0  0.0  0.0  0.9  0.0  0.0  0.0  0.0   0.0
 [6,]  0.0  0.0  0.0  0.0  0.0  0.9  0.0  0.0  0.0   0.0
 [7,]  0.0  0.0  0.0  0.0  0.0  0.0  0.9  0.0  0.0   0.0
 [8,]  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.9  0.0   0.0
 [9,]  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.9   0.0
[10,]  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0   0.9
```

Note that this model is nested within the steppingstone model.

`lrtest` takes six arguments; `H0` and `H1` which specifies the names of the functions specifying migration pattern under the null and the alternative hypothesis, the gene frequency data matrix `p`, optionally a vector `theta` specifying from which parameter values under H_0 to simulate from, and `nboots` specifying the number of bootstrap replicates of the likelihood ratio to compute. After some time (maybe some hours), `lrtest`, as a side-effect, displays a summary of the test results:

```
> test <- lrtest(islandmodel,steppingstone,p,nboots=50)
Likelihood-ratio test summary

Observed value of the test statistic:  26.8696459394272
Number of bootstrap samples:  10
Significance probability:  0
```

In addition, a list containing a number of objects is returned: The `$H0` and `$H1` components contains the lists returned by `fitmodel` when H_0 and H_1 are fitted to the observed data. Thus, for our example data set, the maximum likelihood estimates under the `islandmodel` is stored in

```
> test$H0$par
[1] 0.04816419
```

As we see, under the fitted (false) islandmodel, the rate of migration from the 'mainland' has to be as large as 0.04 to account for the small between-population differentiation generated by the high rate of migration between neighbouring population in the true (steppingstone) model.

Note that, unless the optional argument `theta` is given, `lrtest` simulates the distribution of the likelihood ratio from the maximum likelihood estimates under H_0. However, if there is a large amount of bias in the estimates (see e.g. Tufto et al., 1998), it may be adviceable to first do some sort of bias correction and supply `lrtest` with the bias corrected estimate through the `theta` argument.

The simulated (log) likelihood ratios are stored in the `$lr` component of the list returned by `lrtest`. A histogram of these can be produced with S-Plus' `hist` function (see Figure 1). As can be seen from the figure, the distribution is certainly far from χ^2.

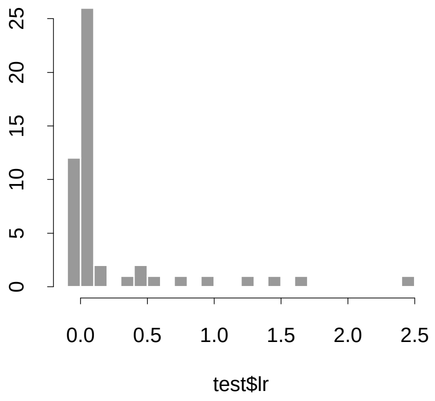

Figure 1: Histogram of simulated log likelihood ratios

Nothing prevents the user from carrying out likelihood ratio tests between non-nested models, although the interpretation of such tests may be questionable from a theoretical point of view (see e.g. Vuong, 1989). It should be kept in mind that such tests can be carried out in two directions. It is therefore up to the user to decide which model represents the null hypothesis and which represent the alternative.

9 Plotting predicted and observed covariances

Once you have fitted a number of alternative models, it's a good idea to inspect how well the model actually fits the data. One way to do this is to compare the covariances between the gene frequencies predicted by the selected model with the covariances computed from the date directly.

The likelihood is computed on the basis of $n-1$ contrasts $\mathbf{y} = (y_1, y_2, \ldots, y_{n-1})$ between the observed standardized gene frequencies. The covariance matrix of these contrasts is returned by function `covpred`. Two arguments are required; the name of the migration model `FUN`, the parameter vector `theta`.

Function `covobs` estimates the covariances from the observations directly using the estimator

$$\hat{c}_{ij} = \frac{1}{n_{loci}} \sum_{k=1}^{n_{loci}} \frac{(p_{i,k} - \bar{p}_k)(p_{j,k} - \bar{p}_k)}{\bar{p}_k(1-\bar{p}_k)}. \tag{4}$$

The only required argument is the matrix of gene frequencies `p`. The covariances can be estimated based on weighted gene frequency means at each loci, by providing a vector `w` of weights to `covobs` as a second optional argument.

The predicted and observed covariances can be compared through plotting them with the `covplot` function. Arguments are the computed predicted and observed covariance matrices. The following call

```
> covplot(covpred(steppingstone,fit1$par),covobs(p))
```

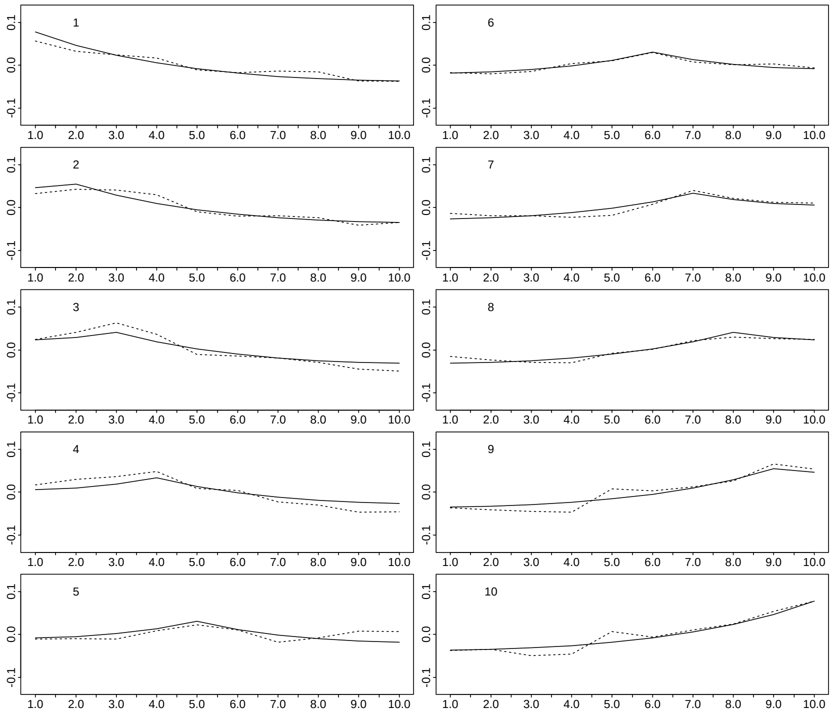

Figure 2: Predicted and observed covariances plotted using the `covplot` function

produced the plot shown in Figure 2. Two additional optional arguments can be given; `mfcol` specifying the number of rows and columns in the plot, and `file` specifying the name of a postscript file (used as alternative output). The interpretation of these type of plots is discussed in Tufto et al. (1998).

Function `covpred` computes (an approximation) of the covariances under a given model, conditioned on the observed gene frequency means at each locus. The unconditional covariances may also be of interest. These can be computed by function `courgeau`, called with the migration matrix `M` and the vector of effective population sizes `Ne` as arguments. This function rewrites the matrix equation (Tufto et al., 1996, eq. 7) to a system of $n(n+1)/2$ equations (Tufto et al., 1996, eq. A.4 and A.5) and solves these. It may be noted that the returned unconditional covariances (which are exact to the extent that the order of the events in the life cycle can be ignored) can differ greatly from the conditional ones, especially if the long range rate of migration is low such that the gene frequency vector is in one of the states $(0,0,\ldots,0)$ or $(1,1,\ldots,1)$ for long periods of time.

10 Sampling error

The model is able to handle most forms of sampling error. Typically, only a proportion of individuals are sampled in each subpopulation. The most realistic assumption is to assume that the number of copies of, say allele A, conditional on the allele frequency p_i in subpopulation i follow a hypergeometric distribution. If we have hypergeometric sampling of $N_i^{(s)}$ individuals from a finite population of $N_i^{(h)}$ individuals, then the conditional variance in the sampled gene frequency is

$$\mathrm{Var}(p_i'|p_i) = \frac{2N_i^{(h)} - 2N_i^{(s)}}{2N_i^{(s)}(2N_i^{(h)} - 1)} p_i(1-p_i). \tag{5}$$

The unconditional standardized variance of the sampled gene frequency is then

$$\mathrm{Var}(p_i) = \frac{1}{2N_i^*} + (1 - \frac{1}{2N_i^*})c_{ii}, \tag{6}$$

where

$$\frac{1}{2N_i^*} = \frac{2N_i^{(h)} - 2N_i^{(s)}}{2N_i^{(s)}(2N_i^{(h)} - 1)}. \tag{7}$$

This simplifies to the more standard binomial model when $N_i^{(h)} \to \infty$. Also, as $N_i^{(s)} \to \infty$ there sampling variance tends to zero.

To set up the model with sampling error, the two vectors `Ns` and `Nh` containing the appropriate sample sizes in each subpopulation (corresponding to $N_i^{(s)}$ and $N_i^{(h)}$, respectively) should be defined globally. The diagonal elements of the computed covariance matrix are then adjusted according to (6), before evaluating the likelihood of the data. In addition, in the `simulate` procedure, sampling error is added at the end of each simulated realization of the process. Sample sizes differing between loci is currently not handled. Assigning the value `Inf` to all elements of `Nh` results in binomial sampling. The model is set up with no sampling error by assigning a `NA` to `Ns` or by assigning `Inf` to some (or all) of the elements.

11 Unsampled subpopulations

Typically, in most studies, sampling only take place in some of the populations, whereas no information is available about the gene frequencies in surrounding subpopulations. One way of handling this is simply to ignore the problem. A more satisfactory solution, perhaps somewhat experimental, is to compute the covariances for the entire population system, including unsampled populations, and then compute the likelihood of the data based on the appropriate covariance submatrix. Similarly, in the simulations, the process of genetic drift in the entire system should be simulated to produce bootstrap replicates of the data in the sampled regions of the population.

To incorporate unsampled subpopulations, `Ne` should be set up contain the effective size of all subpopulations. In addition, the elements of the boolean vector `sampled` should be set up to specify which subpopulations are sampled and which are not, for example:

```
> sampled <- c(F,F,F,T,T,T,T,F,F)
```

Also note that the length of `Ns` and `Nh` should match the number of true elements in `sampled`.

12 Computing the effective size

The eigenvalue variance effective size determining the asymptotic rate of convergence of a subdivided population is determined by eigenvalue of the transition matrix in the recurrence equation for the variances and covariances rearragned into $n(n+1)/2$ column vectors. For details, see Tufto and Hindar (2002). The function `effectpop` returns the effective size based on this approach and takes two arguments: a migration matrix $\mathbb{M}$ and a vector of effective population sizes of each subpopulations. For example, the effective size of population with migration following a stepping stone pattern can be computed as follows:

```
> M <- steppingstone(c(0,.1),n=4)
> M
[,1] [,2] [,3] [,4]
[1,] 0.95 0.05 0.00 0.00
[2,] 0.05 0.90 0.05 0.00
[3,] 0.00 0.05 0.90 0.05
[4,] 0.00 0.00 0.05 0.95
> N <- rep(100,4)
> N
[1] 100 100 100 100
> effectpop(M,N)
[1] 413.0410
```

13 Known bugs

The `bootstrap` and `simulate` functions running under R version 0.62.2, occasionally makes R terminate with the error message "Segmentation fault - core dumped". This is supposedly due to internal bugs in R's memory management. Luckily, this bug appears to have disappeared in 0.63.0 and later versions of R.

14 Alternatives to R and S-Plus

One disadvantage of using R and S-Plus is that these languages are a bit slow. There is probably not much to be gained by rewriting things in an other language, however, since most of the computer time, in any case, will be consumed in matrix manipulations, which, in S-Plus (and I think in R too), are carried out internally with efficient code from the LINPACK library.

15 Things to do

The code is currently only able to handle diallelic loci. With multiallelic loci, say k alleles, one needs the covariances between frequencies of different alleles, between different subpopulations. These are straightforward to derive but the resulting covariance matrices involved would becomes much bigger, and would differ between different loci making the code rather cumbersome to implement. I also haven't figured out how to simulate the conditional distribution conditioned, not on a single allele frequency mean, but on a vector of means $(\bar{p}_1, \bar{p}_2, \ldots, \bar{p}_{k-1})$.

16 Contacting the author

If you download something I would like to hear from you. This way I can keep you informed about future updates etc. In addition, comments, questions, ideas, bug reports, success stories, contribution of additional functions, etc. are highly welcome. My email address is `jarlet@math.ntnu.no`

References

Cabrera, J. and G. S. Watson, 1997. Simulation methods for mean and median bias reduction in parametric estimation. *J. Stat. Plan. Infer.* **57**:143–152.

Efron, B. and R. J. Tibshirani, 1993. An Introduction to the Bootstrap. Chapman & Hall, London.

Felsenstein, J., 1982. How can we infer geography and history from gene frequencies? *Journal of Theoretical Biology* **96**:9–20.

Tufto, J., S. Engen, and K. Hindar, 1996. Inferring patterns of migration from gene frequencies under equilibrium conditions. *Genetics* **144**:1911–1921.

Tufto, J. and K. Hindar, 2002. Effective size in management and conservation of subdivided populations. *Conservation Biology* submitted.

Tufto, J., A. F. Raybould, K. Hindar, and S. Engen, 1998. Analysis of genetic structure and dispersal patterns in a population of sea beet. *Genetics* **149**:1975–1985.

Vuong, Q. H., 1989. Likelihood ratio tests for model selection and non–nested hypotheses. *Econometrica* **57**:307–335.