Tutorial For Using `related`

Last Updated: June 14, 2018

by

Tim Frasier
(timothy.frasier@smu.ca, www.frasierlab.ca)

Contents

1 Brief Description

`related` is an R package that allows users to estimate pairwise relatedness for pairs of individuals based on codominant molecular markers (microsatellites, SNPs, etc.), and also has simulation capabilities for comparing the performance of different estimators and for testing the resolution of a data set. It can also test for relatedness patterns within groups. Relatedness can be estimated using any of seven different methods, and can incorporate inbreeding and genotyping errors. The underlying code for implementing these relatedness estimators is Jinliang Wang's Fortran code for his COANCESTRY program (Wang 2011), and we are thankful to Dr. Wang for allowing us to use his code for this package.

The simulation function, which is new in `related` and is different than that implemented in COANCESTRY, allows users to use an allele frequency file to generate simulated individuals of known relatedness (parent-offspring, full–sibs, half–sibs, and unrelated). The rationale for providing this functionality are two-fold. First, it is well known that some relatedness estimators perform better than others, but this varies depending on the characteristics of the data set (Van de Casteele *et al.* 2001; Csilléry *et al.* 2006; Wang 2011). Thus, users can generate simulated individuals from their allele frequency file, conduct relatedness analyses on those individuals, and then assess which estimator will perform best with their data set.

Secondly, it is often difficult to know what sort of resolution to expect from a data set (e.g. how much differentiation to expect between full- and half-sibs?). Our simulation function provides users with a means to clearly test and visualize what sort of resolution is obtainable with their data.

Lastly, the package also has functionality to test the hypothesis that relatedness within groups is greater than expected if group membership is random with respected to relatedness.

2 Installation

2.1 Setting Up Your Computer

There are a few things to do prior to downloading and installing `related`. First, you obviously need to have R installed on your computer, which can be downloaded from the CRAN (Comprehensive R Archive Network) website: `cran.r-project.org`. I would also highly recommend downloading `RStudio` and working with R through that. `RStudio` can be downloaded from `www.rstudio.com`. You will also need to install the **ggplot2** package, which can be done through R.

2.2 Installing `related`

Most R packages are also available from the CRAN website. Ideally, the `related` package would be hosted there too. However, the original FORTRAN code writes (and then removes) several files to the user's computer throughout the analyses. CRAN has several strict rules that packages must follow in order for them to be hosted there. One of them is that packages cannot write files to user's computers. Therefore, for the moment the package will not be available through CRAN.

Instead the `related` package can be downloaded from our lab website `http://frasierlab.ca` and then clicking on the "Software" link. There, you can download:

1. A tutorial that walks you through all types of analyses with `related`;
2. The R manual for `related`, which is in the "classic" format of R documentation;
3. An example file for testing; and
4. The package itself.

The package is available a binary for Mac OSX, and as source code for Windows and Linux. Thus, clicking on the **download** link will bring you to a GitHub folder where you can download the appropriate file:

- `related_1.0.tgz` for Mac OS X
- `related_1.0.tar.gz` for Windows and Linux

Click on the appropriate file for your operating system, and then click on the "**Raw**" button. This will download the package to your computer.

You can also download the example files ("`GenotypeData.txt`" and "`simrel.csv`") by right-clicking on the file name, and then selecting "**Download Linked File As**.

2.2.1 On A Mac

If you are on a Mac, you can move the package file to R's working directory (where R will look for files). If you don't know where this is, you can open R and type `getwd` and it will give you the path to the current working directory.

Then, you can load the package in one of two ways. The first way is through RStudio.

1. Open RStudio
2. Select **Tools** → **Install Packages** from the main menu
3. In the **Install from:** drop-down menu, select `Package Archive File (.tgz; .tar.gz)`
4. Click on **Browse** and navigate to the folder where the package is located
5. Click on **Install**

The second way is through the command line in R

1. Open R or RStudio
2. In the command line type `install.packages("related_1.0.tgz", repos=NULL, type="mac.binary")`

On a Mac, you may also have to download and install three library files, which can also be downloaded from the same website (to download them, click on the file name and then click on the "Raw" button, and it will automatically download). These are:

1. `libgfortran.3.dylib`
2. `libquadmath.0.dylib`
3. `libgcc_s.1.dylib`

These libraries need to be placed in your `/usr/local/lib` folder. You can get to this folder by opening Finder, and then selecting **Go** → **Go to Folder** and then type `/usr/local/lib/` in the box. Then just move the files there. It will likely ask you for your password. If you already have some of these files there, you should replace the older files with these new ones.

That should do it!

2.2.2 With Windows

To install `related` on a Windows computer, you first need to install the latest version of **Rtools** from `https://cran.r-project.org/bin/windows/Rtools`. This will install the necessary programs and compilers for installing the package in R. Once these are installed, move the package file to R's working directory (where R will look for files). If you don't know where this is, you can open R and type `getwd` and it will give you the path to the current working directory.

Then, you can load the package in one of two ways. The first way is through RStudio.

1. Open RStudio
2. Select **Tools** → **Install Packages** from the main menu
3. In the **Install from:** drop-down menu, select `Package Archive File (.zip; .tar.gz)`
4. Click on **Browse** and navigate to the folder where the package is located
5. Click on **Install**
6. That should do it!

The second way is through the command line in R

1. Open R or RStudio
2. In the command line type `install.packages("related_1.0.tar.gz", repos=NULL, type="source")`
3. . . . and that should do it.

2.2.3 With Linux

To install `related` on Linux, you will need to have a gfortran compiler installed. Most Linux machines will likely already have this installed. If not, the following command should work (from the **terminal** command line **not** the R command line):

```
        sudo apt-get install gfortran
```

Once you have R installed, you may then need to install the R development package, which can be done from the **terminal** with the following command:

```
        sudo apt-get install r-base-dev
```

Once these have been loaded, move the package file to R's working directory (where R will look for files). If you don't know where this is, you can open R and type `getwd` and it will give you the path to the current working directory. There are then two ways to load the package, as with the other two operating systems. The first way is through RStudio.

1. Open RStudio
2. Select **Tools** → **Install Packages** from the main menu
3. In the **Install from:** drop-down menu, select `Package Archive File (.tar.gz)`
4. Click on **Browse** and navigate to the folder where the package is located
5. Click on **Install**

6. That should do it!

The second way is through the command line in R

1. Open R or RStudio
2. In the command line type `install.packages("related_1.0.tar.gz", repos=NULL, type="source")`
3. . . . and that should do it.

2.2.4 Across All Operating Systems

Now, `related` is installed on your computer, and R knows where to find it. However, it is not loaded into R. Therefore, to use `related` (as with any R package), you first need to type

```
> library(related)
```

This loads the package into R's work space, and is required in each R session in which you want to use `related` (i.e. when you close R, it clears your work space, and therefore all desired packages must be reloaded when R is started again).

2.3 Uninstalling `related`

To remove `related` from your computer, from within R type the line below.

```
> remove.packages("related")
```

3 Getting Data Into R In Correctly Formatted Form

3.1 Format

One source of headaches for population geneticists is the various formats of infiles that are required by different analysis programs. Therefore, we have tried to make the required format, and the formatting process, as easy and intuitive as possible.

The format of the genotype file required by `related` is simply one column of individual identifiers, followed by the genotype data, for which there are two columns for each locus (one for each allele). The general rules for the format of the genotype file are:

1. It should be a text file (not an Excel file);
2. It should be space– or tab–delimited;
3. Missing data must be represented by zeros (0) or NA; and
4. There should not be a row of column names in the genotype file.

The example genotype file provided with the package (GenotypeData) can be viewed as an example. The first few lines of that file are represented below. Briefly, each row contains data for one individual. The first column contains text representing individual identifiers for each individual. The second and third columns contain the two alleles for the first locus, columns 4 and 5 contain the two alleles for the second locus, and so on.

AA_n001aripo	241	241	150	150	125	131	...
AA_n002aripo	143	169	198	198	131	131	...
AA_n003aripo	152	169	178	202	131	131	...
...							

Note: if you want to analyze relatedness values based on pre-defined groups (e.g. compare relatedness within versus among groups), then the first two (2) characters of each individual ID should represent the labels for each group. If you look at the example infile (GenotypeData) you will see that this has been done, with the group identifiers being AA, AB, AC, and so on. See section 4.5 for more information.

3.2 Getting Data Into R

The example data file can be accessed in two ways. First, it is already available within the package, and can be loaded into the session using the `data` function:

```
> data(GenotypeData)
```

And then analyses can be conducted using the commands discussed below, such as:

```
> coancestry(GenotypeData, wang=1)
```

Second, the external file can be downloaded from the website, and then loaded into R. We focus on this approach because it is closer to what users will have to do with their own data. There are three ways in which external genotype data can be leaded into R for analysis with `related`:

1. Directly analyze the data via the text file;
2. Use our import function; and
3. Import and format it yourself.

3.2.1 Directly Analyze Text File

First, for calculating relatedness estimates, you can just provide `related` with the name of your genotype file, which needs to be in the format discussed in section 2.1. For this, the genotype file needs to be in R's working directory on your computer (i.e, it must be in the folder where R knows to look for it). Otherwise, you will need to provide the path to your file as well.

For example, if a user wants to get point estimates using Queller & Goodnight's estimator (more on the different relatedness functions later), and their data file is called 'GenotypeData.txt', then the appropriate command would be:

```
> outfile <- coancestry("GenotypeData.txt", quellergt=1)
```

The function for estimating relatedness is called `coancestry`, and here the first argument is the name of the genotype file, in quotes. The second argument indicates that we want point estimates of relatedness using the Queller & Goodnight estimator (more details on the different estimators, and associated options, are in section 3).

3.2.2 Use Our Import Function

To try to make life easier, we have written a function that will read your data into an R data frame in a way that will allow `related` to access it. The function is called `readgenotypedata`. This will create a data frame with your genotype data, and will also create an allele frequency data frame that can be used for other analyses. For example, if your genotype data are in a file called "GenotypeData.txt", then this command would be:

```
    > input <- readgenotypedata("GenotypeData.txt")
```

This will generate a few data frames, all of which are named as an extension to the filename that you specified ("input" in this case), and are read directly from the data (you don't have to enter any information):

`input$gdata`	The data frame containing your genotype data. The first column is character data, and the remaining columns are all integers. This is the same format as the genotype file shown in section 2.1.
`input$nloci`	An integer containing the number of loci used.
`input$nalleles`	A series of integers specifying the number of alleles at each locus.
`input$ninds`	An integer containing the number of individuals in the genotype file.
`input$freqs`	The allele frequency data, which will be useful in other analyses.

You could then conduct the same analysis as above (get point estimates using Queller & Goodnight's estimator) using the following command:

```
    > outfile <- coancestry(input$gdata, quellergt=1)
```

Here, the first argument of the function refers to the data frame containing the genotype data, and the second argument indicates that point estimates using the Queller & Goodnight estimator are to be calculated. The output for this function call is assigned to an R data frame called 'outfile'.

3.2.3 Read in Data Manually

Some users that are familiar with R may prefer to do things their own way. This is possible too.

The general way to read data frame-like data into R is using the R function `read.table`. With this, you have to specify at least two characteristics of your data file: (1) whether or not there is header information (names for each of your fields/columns); and (2) what the different fields (columns) are delimited by (here they should be either tabs ("`\t`") or space (" ")). However, if you do this, R will automatically assign the data type 'Factor' to the individual IDs, rather than character (all genotypes should be integers, which is correct). To change this, you have to add `stringsAsFactors = FALSE` as an argument to the `read.table` function. The full command would be:

```
    > input <- read.table("GenotypeData.txt", header=FALSE, sep=" ", stringsAsFactors=
      FALSE)
```

You can then conduct analyses on this the same way as above (note that with this method you will not get all of the other associated data, such as number of loci, frequencies, etc.):

```
    > outfile <- coancestry(input, quellergt=1)
```

4 Obtaining Estimates of Relatedness

The underlying code for estimating relatedness is the Fortran code for the program COANCESTRY (Wang 2011), and therefore all of the options for that program are available here. We will briefly describe them here, but users should refer to the COANCESTRY paper itself (Wang 2011), as well as the manual for that program for more details regarding the specifics of the calculations.

Briefly, `related` can estimate pairwise relatedness using any of seven possible estimators. These estimators, the command used to refer to them, and the appropriate reference, are listed below.

Command	Name and/or Reference	Column in Result File
dyadml	dyadic likelihood estimator, Milligan (2003)	11
lynchli	Li *et al.* (1993)	7
lynchrd	Lynch and Ritland (1999)	8
quellergt	Queller and Goodnight (1989)	10
ritland	Ritland (1996)	9
trioml	triadic likelihood estimator, Wang (2007)	5
wang	Wang (2002)	6

4.1 Point Estimates of Relatedness

To obtain point estimates of relatedness, use the command names corresponding to the estimator(s) you want to use and set them equal to 1. The "1" tells the program to just calculate point estimates (rather than point estimates *and* 95% confidence intervals). You do not need to have an argument for all estimators: if they aren't listed as arguments to the function call, then it is assumed that you don't want to estimate them.

4.1.1 Non-likelihood Estimators

To obtain relatedness estimates using the 5 non-likelihood methods (`lynchli, lynchrd, quellergt, ritland, wang`), the command would be:

```
> output <- coancestry("GenotypeData.txt", lynchli=1, lynchrd=1, quellergt=1, ritland
    =1, wang=1)
```

There will be 5 resulting data frames, all of which are named as an extension to the data frame name for the output of the function call that you specified ("output" in this case). These data frames remain the same, and have the same structure, regardless of which estimators you choose to estimate. They are:

`output$relatedness`	A data frame containing all pairwise estimates of relatedness. This will always have 11 columns: (1) an integer for the pair number; (2) the ID for individual #1; (3) the ID for individual #2; (4) the group assignment (see section 4.5); (5 - 11) for the 7 relatedness estimators—contain values of 0 for estimators not chosen.
`output$delta7`	A data frame that contains the Δ_7 estimates for the relatedness estimators that use it (trioml, wang, lynchrd, dyadml). This data frame contains one row for each pairwise comparison, and 8 columns: (1) an integer for the pair number; (2) the ID for individual #1; (3) the ID for individuals #2; (4) the group assignment (See section 4.5); and (5 - 8) estimates of Δ_7 for the 4 relevant estimators (trioml, wang, lynchrd, dyadml), with values of 0 for estimators not chosen.
`output$delta8`	A data frame that contains the Δ_8 estimates for the relatedness estimators that use it (trioml, wang, lynchrd, dyadml). This data frame contains one row for each pairwise comparison, and 8 columns: (1) an integer for the pair number; (2) the ID for individual #1; (3) the ID for individuals #2; (4) the group assignment (See section 4.5); and (5 - 8) estimates of Δ_7 for the 4 relevant estimators (trioml, wang, lynchrd, dyadml), with values of 0 for estimators not chosen.
`output$inbreeding`	A data frame that contains the inbreeding estimates for each individual, as used in the relatedness estimators. Only four of the relatedness estimators can account for inbreeding: dyadml, lynchrd, ritland, trioml. This data frame contains one row for each individual, and 5 columns: (1) individual ID; (2-5) inbreeding estimates for the four relatedness estimators. Estimators not used will have a zero (0) in the corresponding column.

4.1.2 Likelihood Estimators

Note that the likelihood estimators are **much** slower than the others!

Obtaining point estimates for the likelihood estimators (dyadml and trioml) follows the same approach as with the non-likelihood estimators. However, the triadic likelihood estimator (**trioml**) requires that you specify the number of reference individuals to use for estimating relatedness. The default is 100, and therefore if you want to use 100 then you do not need to specify a value. However, if you want to change this (say to 150), then you would have to add the argument `trioml.num.reference = 150` to the function call.

For example, to obtain point estimates for both likelihood estimators, and the wang estimator, but use 150 reference individuals for the trioml estimator, the command would be:

```
> output <- coancestry("GenotypeData.txt", dyadml=1, trioml=1, wang=1, trioml.num.
    reference=150)
```

4.2 Point Estimates of Relatedness With 95% Confidence Intervals

`related` can estimate 95% confidence intervals for relatedness estimates based on bootstrapping over loci. This option can be selected by setting the appropriate relatedness estimator to 2. For example, if you want to estimate relatedness using the Queller and Goodnight (1989), Lynch and Ritland (1999), and Wang (2002) estimators, but also want 95% confidence intervals for the Wang (2002) estimator, the command would be:

```
> output <- coancestry("GenotypeData.txt", quellergt=1, lynchrd=1, wang=2)
```

Or, if you want 95% confidence intervals for all three, then the command would be:

```
> output <- coancestry("GenotypeData.txt", quellergt=2, lynchrd=2, wang=2)
```

The default number of bootstrap replicates is set to 100. However, this can be changed using the `ci95.num.bootstrap` argument in the `coancestry` command. For example, if you want to change the number of bootstrap events to 500, the command would be:

```
> output <- coancestry("GenotypeData.txt", quellergt=2, lynchrd=2, wang=2, ci95.num.
    bootstrap=500)
```

In addition to the output data frames described in section 3.1.1, setting the estimator argument to 2 generates the additional output data frames, which are also named as an extension to the name that you specified ("output" in this case).

`output$relatedness.ci95`	A data frame containing the low and high cut-off values for the 95% confidence interval associated with each chosen estimator. This will always have 18 columns: (1) an integer for the pair number; (2) the ID for individual #1; (3) the ID for individual #2; (4) the group assignment (see section 4.5); (5 - 18) for the high and low values associated with each of the 7 relatedness estimators—contain values of 0 for estimators not chosen.
`output$delta7.ci95`	A data frame that contains the low and high cut-off values for the 95% confidence interval for the Δ_7 estimates associated with each chosen estimator that use it (trioml, wang, lynchrd, dyadml). This will always have 12 columns: (1) an integer for the pair number; (2) the ID for individual #1; (3) the ID for individual #2; (4) the group assignment (see section 4.5); (5 - 12) for the high and low values associated with each of the 7 relatedness estimators—contain values of 0 for estimators not chosen.
`output$delta8.ci95`	A data frame that contains the low and high cut-off values for the 95% confidence interval for the Δ_8 estimates associated with each chosen estimator that use it (trioml, wang, lynchrd, dyadml). This will always have 12 columns: (1) an integer for the pair number; (2) the ID for individual #1; (3) the ID for individual #2; (4) the group assignment (see section 4.5); (5 - 12) for the high and low values associated with each of the 7 relatedness estimators—contain values of 0 for estimators not chosen.
`output$inbreeding.ci95`	A data frame that contains the low and high cut-off values for the 95% confidence interval for the inbreeding estimates for each individual, as used in the relatedness estimators. Only four of the relatedness estimators can account for inbreeding: dyadml, lynchrd, ritland, trioml. This data frame contains one row for each individual, and 9 columns: (1) individual ID; (2-9) inbreeding estimates for the four relatedness estimators. Estimators not used will have a zero (0) in the corresponding column.

4.3 Including Genotyping Error Rates

To account for genotyping error rates, the `error.rates` argument can be added to the initial call of the function. A single value can be entered if the error rates are the same across all loci, or a different value can be entered for each locus. For example, to estimate relatedness using the Wang (2002) estimator, using an error rate of 0.01 for all loci, the command would be:

```
    > output <- coancestry("GenotypeData.txt", error.rates=0.01, wang=2)
```

If different error rates are desired for different loci, then a vector can be made of the error rates for each locus, and then this vector can be referred to by the `error.rates` argument. An example is show below assuming genotypes at 10 loci. The "`c`" in the function below stands for "concatenate", which is an R command for concatenating values into a single variable.

```
    > errors <- c(0.01, 0.02, 0.03, 0.04, 0.05, 0.06, 0.07, 0.08, 0.09, 0.10)
    > output <- coancestry("GenotypeData.txt", error.rates=errors, wang=2)
```

Note that including error rate estimates does not change point estimates of relatedness, just the confidence intervals.

4.4 Accounting For Inbreeding

The two likelihood methods, lynchrd, and ritland can account for inbreeding in their estimates of relatedness. This option can be selected using the `allow.inbreeding` argument. The default for this is FALSE, but it can be changed to TRUE if you want to account for inbreeding in your estimates. For example, to estimate relatedness using the dyadml estimator, and allow for inbreeding, while also calculating confidence intervals, the command would be:

```
    > output <- coancestry("GenotypeData.txt", allow.inbreeding=TRUE, ci95.num.bootstrap
        =100, dyadml=2)
```

Note that accounting for inbreeding greatly increases the time needed for the program to run.

5 Conducting Simulations

Simulations can be conducted using the `familysim` function. This function requires two arguments: (1) the name of the allele frequency data; and (2) the number of individuals to simulate. Both the `readgenotypedata` and `coancestry` functions generate allele frequency data in a format that can be used for simulations. Only one number should be provided for the number of individuals to simulate, and this represents the number of pairs to generate for each degree of relatedness. For example, entering `100` would generate 100 parent-offspring pairs, 100 full-sib pairs, 100 half-sib pairs, and 100 unrelated pairs.

As an example, the commands for reading genotype data into R, and generating 100 simulated pairs of individuals of each degree of relatedness would be:

```
  > input <- readgenotypedata("GenotypeData.txt")
  > sim <- familysim(input$freqs, 100)
```

The result is a data frame of simulated genotypes that can be analyzed using the `coancestry` function. For example, to analyze these simulated data using the Wang (2002) estimator the code would be:

```
> output <- coancestry(sim, wang=1)
```

One issue with the way this works is that `coancestry` will calculate relatedness for **all** pairwise comparisons of the simulated individuals, whereas we are only interested in the relatedness values for the specific pairs that we generated of known relatedness. For example, for parent–offspring we are only interested in the relatedness of individuals 1 & 2, 3 & 4, 5 & 6, and so on. We don't care about the relatedness of 1 & 3, 1 & 4, etc. To deal with this, we have created a function called `cleanuprvals` that will remove the unwanted relatedness values. This function takes two arguments: (1) the name of the data frame containing relatedness values that will be "trimmed"; and (2) the number of individuals of each pair that were simulated. Thus, for this example the command would be:

```
> simrel <- cleanuprvals(output$relatedness, 100)
```

Now, you can analyze the relatedness values of these simulated individuals of known relatedness in whatever way is most informative for you and best suits your goals. For example, you may want to plot the distribution of relatedness values to see how well you can discriminate between individuals of different relatedness. There are two primary ways to do this: with box plots, or with density graphs. We'll walk through both of them.

5.1 Box Plots

Prior to plotting the data, it is helpful to clean the data up further. What we will do here is make new data frames consisting of two types of data: labels for each degree of relatedness, and the relatedness values. The Wang (2002) estimator is in the 6th column of the results file, so we can place that into a new list using the following command:

```
> relvalues <- simrel[, 6]
```

Using this sort of syntax in R, you can select specific rows and columns that you want to select from a given data frame. The row selections are indicated before the comma, and the column selections are indicated after the comma. If you don't include a number, this means "select all". Therefore, our command above selects all rows, and only the 6th column of the simrel data frame.

Now we need to add a label indicating the degree of relatedness of each pair. We can do this with the following commands:

```
> label1 <- rep("PO", 100)
> label2 <- rep("Full", 100)
> label3 <- rep("Half", 100)
> label4 <- rep("Unrelated", 100)
```

These commands make use of the "repeat" command in R, which is abbreviated to `rep`, which will generate a list containing the specified value repeated the specified number of times. So the first command is making a list, called "label1" that contains the text "PO" repeated 100 times, and so on. It is very important that these are made in this order, because this corresponds to the order of relatedness of the simulated pairs. We can now combine these into one long list using the following command:

```
> labels <- c(label1, label2, label3, label4)
```

The `c` in the above command stands for "concatenate", and so the above command is concatenating the four lists into one long list.

To plot a box plot, you have to tell R what list of data you want to group the data by, and this is called the `factor`. Here, we want to group the data by the labels indicating the degree of relatedness. We can specify which field to use for grouping values by with the `as.factor` function. The command is below (results in **Figure 1**).

```
> plot(as.factor(labels), relvalues, ylab="Relatedness Value", xlab="Relatedness")
```

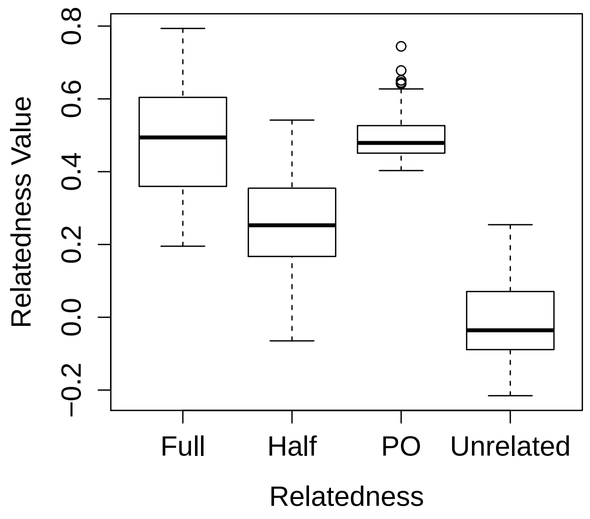


Figure 1: Box plot of relatedness values for simulated pairs of known relatedness using the `plot` function.

This looks good, but I prefer the plots that are generated with the **ggplot2** package, is installed automatically when you load `related`. To generate a similar plot, use the following commands (see **Figure 2**):

```
> qplot(as.factor(labels), relvalues, geom="boxplot", ylab="Relatedness Values", xlab="
    Relatedness")
```

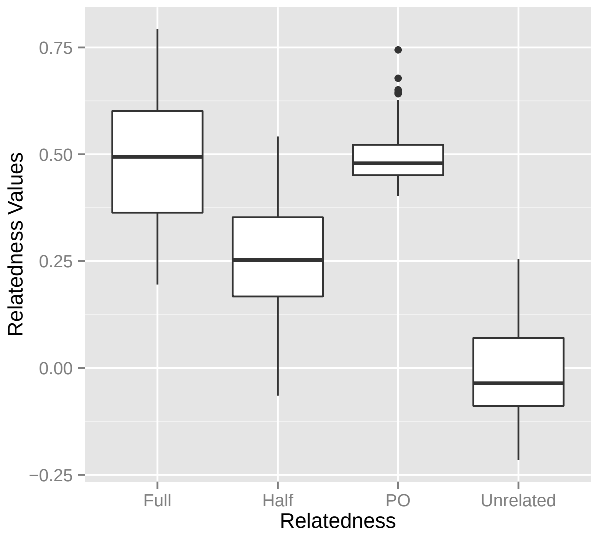


Figure 2: Box plot of relatedness values for simulated pairs of known relatedness using ggplot2.

5.2 Density Plots

Another informative way to assess the overlap between relatedness values of individuals of different relatedness is to create density plots representing histograms of the relatedness values. This is also easiest to do with the `ggplot2` package.

We will need to do a few things to get the data ready for plotting. First, let's rename the list of labels, because this name will be used in the legend of our graph.

```
> Relationship <- labels
```

Next, we need to combine the `Relationship` and `relvalues` vectors into a single data frame. We can do this with the `cbind` function in R, which is for binding columns together. We also want to make sure the resulting object is saved as a data frame.

```
> newdata <- as.data.frame(cbind(Relationship, relvalues))
```

The above command codes the `relvalue` data in the wrong format, and we need to correct that.

```
> newdata$relvalues <- as.numeric(as.character(newdata$relvalues))
```

Now we can plot a density plot of the relatedness values for each type of relationship (see **Figure 3**).

```
> qplot(relvalues, ..density.., data=newdata, geom="density", colour=as.factor(
    Relationship), xlab="Relatedness Value", ylab="Density")
```

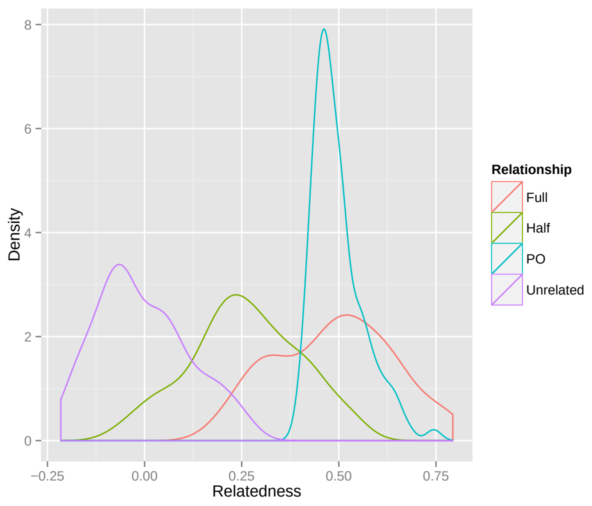


Figure 3: Density plot of relatedness values for simulated pairs of known relatedness using ggplot2.

5.3 Comparing Relatedness Estimators

Prior to deciding which relatedness estimator to use for your analysis, it is a good idea to test the performance of the different estimators on simulated data sets with the same locus characteristics as your own. There are two ways in which you can do this. First, we have created a function that will automate this process for you, called `compareestimators`, which compares the lynchli, lynchrd, quellergt, and wang estimators. Second, if you want to compare different estimators, the commands for doing so are provided below.

5.3.1 Using `compareestimators` Function

This function takes two arguments: (1) the name of the data frame containing your genotype data; and (2) the number of simulated pairs of individuals you want for each type of relationship. This function will generate simulated genotypes of known relatedness, calculate relatedness values using 4 of the moment-based estimators, and then plot the data for your visualization. The Ritland (1996) estimator is not included in this comparison because we found that the variation around these estimates were vastly larger than for the other estimators, which made the required scales for the estimators very different, and therefore compromised the usefulness of the subsequent graphs.

For example, using the example file the commands would be:

```
> input <- readgenotypedata("GenotypeData.txt")
> compareestimators(input, 100)
```

The output will be a graph of the data, organized by relationship type and estimator (see **Figure 4**). Note that it will take a while to do all of the analyses and show the graph. Be patient!

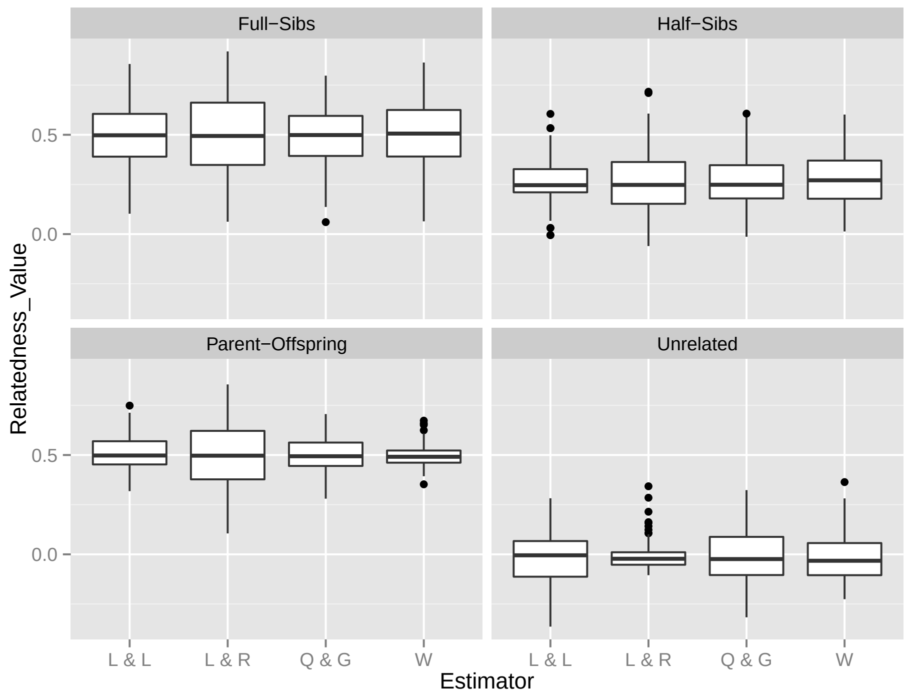


Figure 4: Plot comparing the relatedness estimates using 4 different estimators, and simulated individuals of known relatedness. "L & L" is the lynchli estimator, "L & R" is the lynchrd estimator, "Q & G" is the quellergt estimator, and "W" is the wang estimator.

The program will also estimate the correlation coefficient ("Pearson's") between observed and expected relatedness values for each estimator, and print these to the screen. Based on these values you can see which estimator correlates best with expected values.

5.3.2 Custom Comparisons

It is also relatively easy to compare a different set of relatedness estimators. Suppose you want to compare the two likelihood estimators with those of Queller & Goodnight and Wang. First, load the data into R and generate as many simulated individuals as you like (here we'll use 100).

```
> input <- readgenotypedata("GenotypeData.txt")
> simdata <- familysim(input$freqs, 100)
```

Then, estimate relatedness using the desired estimators (note that this will take a long time if using the likelihood estimators).

```
> output <- coancestry(simdata, dyadml=1, trioml=1, quellergt=1, wang=1)
```

This file will contain *all* pairwise estimates of relatedness, rather than just the ones that we are interested in. To reduce this to only the desired values, use the `cleanuprvals` function.

```
> simrel <- cleanuprvals(output$relatedness, 100)
```

Next, we need to parse out the data based on relatedness type and estimator used. In the file, the first 100 pairs are parent-offspring, the second set of 100 are full-sibs, the third set of 100 are half-sibs, and the fourth (last) set of 100 are unrelated. Obviously, these numbers would change if you simulated a different number of individuals. Moreover, the trioml estimates are in column 5, the wang estimates are in column 6, the quellergt estimates are in column 10, and the dyadml estimates are in column 11 (see table on p. 8). What the code below is doing is selecting the range of rows and columns that correspond to the appropriate relatedness value and estimator.

```
> triomlpo <- simrel[1:100, 5]
> triomlfs <- simrel[(100 + 1) : (2 * 100), 5]
> triomlhs <- simrel[((2 * 100) + 1) : (3 * 100), 5]
> triomlur <- simrel[((3 * 100) + 1) : (4 * 100), 5]
> wangpo <- simrel[1:100, 6]
> wangfs <- simrel[(100 + 1) : (2 * 100), 6]
> wanghs <- simrel[((2 * 100) + 1) : (3 * 100), 6]
> wangur <- simrel[((3 * 100) + 1) : (4 * 100), 6]
> quellergtpo <- simrel[1:100, 10]
> quellergtfs <- simrel[(100 + 1) : (2 * 100), 10]
> quellergths <- simrel[((2 * 100) + 1) : (3 * 100), 10]
> quellergtur <- simrel[((3 * 100) + 1) : (4 * 100), 10]
> dyadmlpo <- simrel[1:100, 11]
> dyadmlfs <- simrel[(100 + 1) : (2 * 100), 11]
> dyadmlhs <- simrel[((2 * 100) + 1) : (3 * 100), 11]
> dyadmlur <- simrel[((3 * 100) + 1) : (4 * 100), 11]
```

Next, we need to create a list of labels for the different estimators, with each repeated the appropriate number of times (100 in this case).

```
> trioml <- rep("tri", 100)
> wang <- rep("W", 100)
> quellergt <- rep("QG", 100)
> dyadml <- rep("di", 100)
> estimator2 <- c(trioml, wang, quellergt, dyadml)
> Estimator <- rep(estimator2, 4)
```

Create a list of labels for the different relatedness types

```
> po <- rep("Parent-Offspring", (4 * 100))
> fs <- rep("Full-Sibs", (4 * 100))
> hs <- rep("Half-Sibs", (4 * 100))
> ur <- rep("Unrelated", (4 * 100))
> relationship <- c(po, fs, hs, ur)
```

Combine the different values for each estimator based on relatedness type, as lists.

```
> relatednesspo <- c(triomlpo, wangpo, quellergtpo, dyadmlpo)
> relatednessfs <- c(triomlfs, wangfs, quellergtfs, dyadmlfs)
> relatednesshs <- c(triomlhs, wanghs, quellergths, dyadmlhs)
> relatednessur <- c(triomlur, wangur, quellergtur, dyadmlur)
> Relatedness_Value <- c(relatednesspo, relatednessfs, relatednesshs, relatednessur)
```

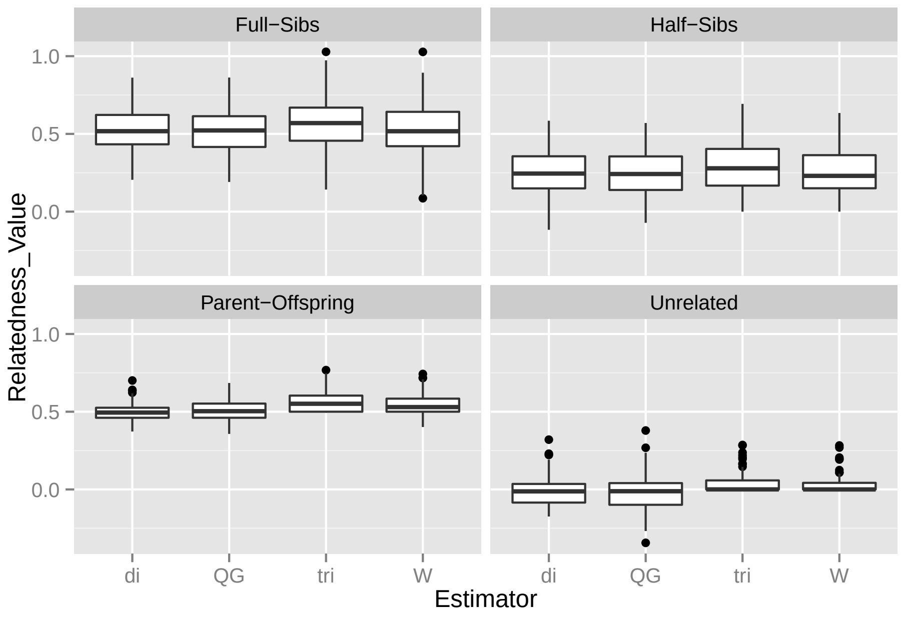
Combine the data.

```
> combineddata <- as.data.frame(cbind(Estimator, relationship, Relatedness_Value))
> combineddata$Relatedness_Value <- as.numeric(as.character(
    combineddata$Relatedness_Value))
```

Plot the data. You may need to play around with the range of values of the y-axis (the `ylim` argument) so that it better captures your data.

```
> ggplot(combineddata, aes(x = Estimator, y = Relatedness_Value), ylim = c(-0.5,
    1.0)) +
> geom_boxplot() +
> facet_wrap(~ relationship)
```

The resulting plot is shown below. Note that you are not limited to comparing only four estimators. You can compare as many as you wish, just make sure that the commands are changed accordingly.



You can also calculate the correlation coefficient between the observed values for each estimator and the expected values. To do this, first create vectors containing the appropriate relatedness values the appropriate number of times (in this case, 100 times).

```
> urval <- rep(0, 100)
> hsval <- rep(0.25, 100)
> fsval <- rep(0.5, 100)
> poval <- rep(0.5, 100)
> relvals <- c(poval, fsval, hsval, urval)
```

Now you can just compare the observed and expected values, selecting the appropriate column from the `simrel` data data frame for each estimator (the trioml estimates are in column 5, the wang estimates are in column 6, the quellergt estimates are in column 10, and the dyadml estimates are in column 11 (see table on p. 8)).

```
> cor(relvals, simrel[, 5])
> cor(relvals, simrel[, 6])
> cor(relvals, simrel[, 10])
> cor(relvals, simrel[, 11])
```

6 Analyzing Relatedness Within Groups

Sometimes it is desirable to test if individuals within groups are more related than expected if these just represent random groups of individuals. `related` has a function for testing this, called `grouprel`. What this function will do is calculate the average relatedness within each of the specified groups. It will then generate a distribution of 'expected' relatedness values by randomly shuffling individuals between groups, while keeping each group size constant, and calculating the average relatedness within each group for each randomization step. It will then generate plots comparing the observed and expected values. It also writes a file called "`observed-r.csv`", and another one called "`expectedrel.csv`" to your working directory, containing the observed and expected values, respectively, so that you can conduct further analyses of these data.

6.1 Getting Started

For this function to work, you have to code your individual (or sample) identifiers so that **the first two letters** denote the group name. It is these two characters that the function will use to group individuals together!! In the original example file ("GenotypeData.txt"), these are "AA", "AB", "AC", etc.

Let's explore the details by walking through an example. I have made an example file (called "`simrel.txt`", that you can download) containing genotypes for 30 individuals. The first 10 individuals are full-siblings, the second 10 individuals are half-sibs, and the last 10 individuals are unrelated. The "groups" in this case are denoted by the first two letters of the sample names being designated as "FS", "HS", or "UR". First let's read this file into R, using the `readgenotypedata` function.

```
> reldata <- readgenotypedata("simrel.txt")
```

We can then conduct the group analysis using the following command.

```
> grouprel(genotypes = reldata$gdata, estimatorname = "wang", usedgroups = "all",
    iterations = 100)
```

As you can see, this function takes four arguments. We'll walk through each of these in detail.

The first argument is telling the function what genotype data to use for the analyses. Note that this is *just* the genotype data, not the entire set of files generated when data are read into R. Therefore, we have to specify this by using the "`$gdata`" part of our original file.

The second argument tells R what estimator to use. You need to specify this using the names (in quotes) as designated in the table on p. 8. Only one estimator can be used at a time.

The third argument tells R what groups to consider in the analyses. This is based on the two-character code that you have used in your individual/sample names. Here, we have selected "all". However, we could have selected to analyze just the full- and half-sib groups by typing "`usedgroups = c("FS", "HS")`" here. You can use a similar approach for analyzing only certain groups in your data.

The last argument tells R how many times you want to shuffle individuals between groups. Here, we have just chosen 100, but you should do more than this when analyzing data for real. Note that this can take a long time - I mean really long in some cases. This is because `related` recalculates relatedness for each iteration. This means that if it took `related`, say, 3 minutes to estimate relatedness for your observed groups, then if you conduct 1,000 iterations, this will take: 3 minutes $\times$ 1,000 = 3,000 minutes = 50 hours! So, just keep this in mind when you're planning your analyses.

6.2 What the Function Does

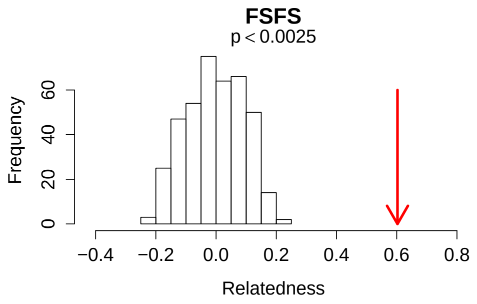
Once you enter the command above, the function will first estimate all pairwise relatedness values *within each group*, and take their average. It will print these averages to the screen (but they may pass by too quickly to be seen), and will also write them to a file in R's working directory called "`observed-r.csv`". This file contains three columns. The first is just a list of integers, indicating how many groups were analyzed, and giving each group an incremental number. The second contains a list of each group "name", where the name is based on the two-character code from the sample/individual identifiers. These characters will be repeated. For example, our groups were designated "FS", "HS", and "UR" in the example file, so the list here will read "FSFS", "HSHS", and "URUR". This just indicates that the values represent relatedness values between individuals designated "FS" and "FS", "HS" and "HS", and "UR" and "UR".

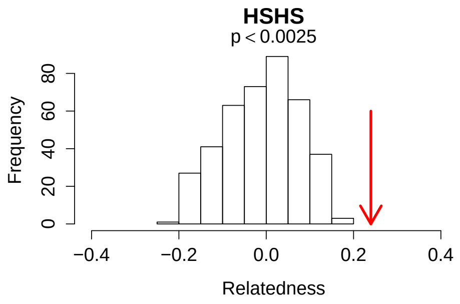
Next, the function will randomly shuffle the order of individuals in the genotype file, but keep the rows representing each group constant (and the same as the observed data). Thus, individuals are shuffled between groups, while keeping each group size constant. Then relatedness is calculated within each group as described above. This process is repeated however many times is designated in the command.

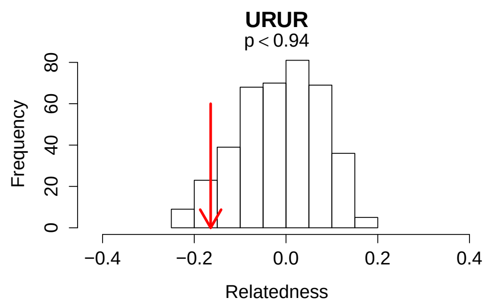
The results are exported to a file called "`expectedrel.csv`". This file will contain one column for each group in the analyses, and one row for each iteration performed (the columns are not labeled, however). Thus, each column represents the distribution of expected relatedness values within that group if group membership is random with respect to relatedness. The order of groups (i.e., the order of columns) is the same order as they are encountered in the genotype file (and as printed in the "`observed-r.csv`" file). The last column is the average relatedness value across all groups for each iteration. This can be used to test global patterns across your data, rather than just for each group independently. By having this as a separate file, you can explore the data on your own.

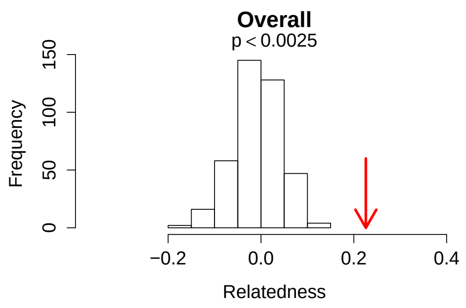
Figures are also generated based on these data. One figure is generated for each analyzed group. Each figure contains: (1) a histogram of the expected relatedness values within each group; (2) a red arrow indicating where the observed value lies, for easy comparison of observed and expected values; (3) a title indicating the group comparison; and (4) a p-value, indicating the percentage of randomized iterations where the expected values were greater than or equal to the observed value (see below for more explanation). A similar figure is generated for the "overall" data set, where the observed average within-group relatedness across all groups is compared to the expected values. Thus, this function simultaneously assess whether or not relatedness *within each group* is higher than expected, and whether or not relatedness *within all groups* is high than expected. P-values cannot be exact in this case, because they are based on simulation rather than direct calculation. Suppose that we conducted 500 simulations, and only had 1 of these simulations with an average within-group relatedness greater than or equal to our observed value. It would be tempting to say that our p-value is 0.002 (1 out of 500). However, we need to be conservative here, because our estimate is based on simulation. The appropriate way to do this is to say that we have observed *fewer than 2 out of 500 values* less than or equal to our observed value. Thus $p < 0.004$ (fewer than 2 out of 500). This is how p-values are reported in these figures.

Examples of these figures, based on 400 iterations and the example file, are below.









6.3 Some Words of Caution

The shuffling process implemented here is an appropriate method to test hypotheses only when the number of possible combinations of your data greatly outnumber the number of iterations that are performed. If the number of iterations you perform outnumbers the possible combinations of your data then you will get the same scenario replicated multiple times in your simulations, and thus get erroneous estimates of your p-values.

As an example, suppose that I have genotyped 100 individuals. Within that data set I have identified two groups of individuals, each group containing three individuals, and the individuals within these groups are always found together. I suspect that they are relatives, and I want to use `related` to test if they are more/-less related than expected based on chance. If I just use the data from the two groups ($N = 6$ individuals) then there are only 10 different ways to group them into two groups of three. Therefore, if I perform 1000 iterations I will have repeated each scenario ∼100 times, making my estimated p-value completely bogus. What would be appropriate in this case would be to make a third "dummy" group containing the other 94 genotypes. This way individuals can be shuffled to and from this dummy group throughout the iterations, and you can get reliable p-value estimates from the simulations for your two groups of interest (because the number of ways to group 6 individuals into two groups of three out of a pool of 100 individuals is much much greater than 1000). Indeed, this is the best way to test the null hypothesis that the two observed groups of three represent random groups of individuals (from your sampled population) with respect to relatedness.

The take home message is to consider your sample size when conducting your analyses and deciding how many shuffling steps to perform for hypothesis testing. Ensure that you have your analyses set up in such a way that the randomization processes will properly test your null hypothesis and will not result in duplicates during the simulation processes.

References

Csilléry K, Johnson T, Beraldi D, Clutton-Brock T, Coltman D, Hansson B, *et al.* (2006). Performance of marker-based relatedness estimators in natural populations of outbred vertebrates. *Genetics* **173**: 2091–2101.

Li CC, Weeks DE, Chakravarti A (1993). Similarity of DNA fingerprints due to chance and relatedness. *Human Heredity* **43**: 45–52.

Lynch M, Ritland K (1999). Estimation of pairwise relatedness with molecular markers. *Genetics* **152**: 1753–1766.

Milligan BG (2003). Maximum-likelihood estimation of relatedness. *Genetics* **163**: 1153–1167.

Queller DC, Goodnight KF (1989). Estimating relatedness using molecular markers. *Evolution* **43**: 258–275.

Ritland K (1996). Estimators for pairwise relatedness and inbreeding coefficients. *Genetical Research* **67**: 175–186.

Van de Casteele T, Galbusera P, Matthysen E (2001). A comparison of microsatellite-based pairwise relatedness estimators. *Molecular Ecology* **10**: 1539–1549.

Wang J (2002). An estimator for pairwise relatedness using molecular markers. *Genetics* **160**: 1203–1215.

Wang J (2007). Triadic IBD coefficients and applications to estimating pairwise relatedness. *Genetical Research* **89**: 135–153.

Wang J (2011). COANCESTRY: a program for simulating, estimating and analysing relatedness and inbreeding coefficients. *Molecular Ecology Resources* **11**: 141–145.