# CONGENR: USER MANUAL

ROBERT C. LONSINGER[1,2] AND LISETTE P. WAITS[1]

[1]UNIVERSITY OF IDAHO, DEPARTMENT OF FISH AND WILDLIFE SCIENCES
875 PERIMETER DRIVE, MS 1136, MOSCOW, ID 83844
[2]LONS1663@VANDALS.UIDAHO.EDU

November 4, 2015

**Recommended citation:** Lonsinger RC, Waits LP (2016) ConGenR: rapid determination of consensus genotypes and estimates of genotyping errors from replicated genetic samples. Conservation Genetics Resources.

**Maintainer:** Robert Lonsinger (Lons1663@vandals.uidaho.edu)

**Available at:** www.uidaho.edu/cnr/research-outreach/facilities/leecg/publications-and-software

**Dependencies:** stringr; CONGENR will install and load the R stringr package if it is not already present in your R working directory.

**Description:** CONGENR is a conservation genetics R (R Core Team 2015) script that facilitates the rapid determination of consensus genotypes from replicated genetic samples, determines overall and individual sample level polymerase change reaction (PCR) success rates, and calculates observed genotyping error rates (i.e., allelic dropout [ADO] and false allele [FA] rates). CONGENR allows users to efficiently calculate and compare PCR success rates and genotyping errors by sample classes, which can constitute any identifiable subdivision of samples (e.g., based on sample age or condition, region, season, sex, age class). CONGENR provides estimates of PCR success rates and genotyping error rates by locus, which can expedite the identification of problematic loci (e.g., loci experiencing low PCR success rates and/or high genotyping error rates). Additionally, CONGENR supplies a summary of the number loci that have achieved a consensus genotype for each sample, facilitating the identification of samples requiring additional replicates. CONGENR is intended for use with codominant, multilocus microsatellite data generated primarily through noninvasive genetic samples and processed with a multi-tubes approach (Taberlet et al. 1996). The wrapper function, `ConGen()`, is designed to handle input in a format that can be easily exported from the program GENEMAPPER and calls all the necessary supporting functions to perform the analysis. Researchers interested in calculating genotyping error rates by comparing low quality samples (e.g., noninvasively collected samples) to high quality reference samples (e.g., multilocus genotypes resulting from blood or tissue samples; in place of consensus genotypes) can do so by directly calling the genotyping error function, `gt.error()`; this may be particularly useful when conducting pilot studies to evaluate genotyping error rates using noninvasive samples collected from known individuals from which high quality samples have been obtained. Additionally, CONGENR offers a function to compare multilocus consensus genotypes across samples and identify samples that match at all or a specified number of loci. This function, `congen.matching()`, must be called separately and takes the consensus genotype files produced by `ConGen()` as the input.

*Consensus Genotypes*.—CONGENR employs common protocols for determining consensus genotypes from replicated (muli-tubes approach) DNA samples (e.g., Frantz et al. 2003, Flagstad et al. 2004, Zhan et al. 2010). Specifically, CONGENR requires that each allele of a heterozygous genotype be observed ≥2 times, while single alleles must be observed ≥3 times to confirm a homozygous genotype. Consensus genotypes may include homozygotes with uncertainty, which are characterized by the confirmation of one allele observed 2 – 4 times, while a second allele is observed only once, generating uncertainty as to whether or not this single observation is the result of a single FA, or repeated ADO across the other replicates. Upon completion of additional replicates, if the single occurrence allele is observed a second time, it is then confirmed and the genotype is determined to be heterozygous. Alternatively, if additional replicates beyond the 5$^{th}$ replicate fail to detect the single occurrence allele a second time, the genotype is determined to be homozygous. CONGENR only accepts numeric values for scored alleles and is not set to handle letter or characters (e.g. ?, X, and Y).

*Consensus Genotype Examples*.—The following examples demonstrate the process taken by CONGENR to confirm (or establish) consensus genotypes.

Example 1:

| | | |
|---|---|---|
| Replicate 1 – | 144/148 | Two alleles present/observed |
| Replicate 2 – | 144/148 | Two alleles present/observed |
| Consensus – | 144/148 | Only 2 alleles seen, each seen ≥2x<br>Consensus genotype = Heterozygous |

Example 2:

| | | |
|---|---|---|
| Replicate 1 – | 144/144 | One allele present/observed |
| Replicate 2 – | 144/144 | One allele present/observed |
| Replicate 3 – | 144/144 | A third replicate is required to confirm |
| Consensus – | 144/144 | Only 1 allele seen ≥3x<br>Consensus genotype = Homozygous |

Example 3:

| | | |
|---|---|---|
| Replicate 1 – | 144/148 | Two alleles present/observed |
| Replicate 2 – | 144/144 | One allele previously observed |
| Replicate 3 – | 144/148 | Presence of both alleles in third replicate |
| Consensus – | 144/148 | Two alleles seen ≥2x across replicates<br>Consensus genotype = Heterozygous |

Example 4:

| | | |
|---|---|---|
| Replicate 1 – | 144/148 | Two alleles present/observed |
| Replicate 2 – | 144/144 | One allele previously observed |
| Consensus – | 144/0 | One allele seen 2–4x, second allele seen only once<br>Consensus genotype = Homozygous with uncertainty (0) |

Example 5:

| | | |
|---|---|---|
| Replicate 1 – | 144/144 | One allele present/observed |
| Replicate 2 – | 144/144 | One allele present/observed |
| Replicate 3 – | 148/148 | One (different) allele present/observed |
| Replicate 4 – | 148/148 | One (different) allele present/observed |
| Consensus – | 144/148 | Only 2 alleles seen, each seen ≥2x<br>Consensus genotype = Heterozygous |

Example 6:

| | | |
|---|---|---|
| Replicate 1 – | 144/144 | One allele present/observed |
| Replicate 2 – | 144/144 | One allele present/observed |
| Replicate 3 – | 144/148 | Two alleles present/observed |
| Replicate 4 – | 144/144 | One allele present/observed |
| Consensus – | 144/0 | One allele seen 2–4 x, second allele seen only once<br>Consensus genotype = Homozygous with uncertainty (0) |

Example 7:

| | | |
|---|---|---|
| Replicate 1 – | 144/144 | One allele present/observed |
| Replicate 2 – | 144/144 | One allele present/observed |
| Replicate 3 – | 144/148 | Two alleles present/observed |
| Replicate 4 – | 144/144 | One allele present/observed |
| Replicate 5 – | 144/144 | One allele present/observed |
| Consensus – | 144/144 | One allele seen ≥5x, second allele seen only once<br>Consensus genotype = Homozygous |

*PCR Success Rates*.—CONGENR calculates PCR success rates based on two approaches.

1. PCR Success – an overall assessment of PCR success, which calculates the number of successful amplification / the total number of amplifications attempted.
2. Individual Sample Level PCR Success – an individual sample level success rate, which calculates the proportion of samples that had successful amplifications at *n* loci. When the argument PID = NULL, *n* = 50% of the loci. Otherwise, PID may be set to any value between 1 and the total number of loci and *n* = PID. In practice, PID will most appropriately be set to the probability of identity (Waits et al. 2001) required for successful identification of individuals.

Additionally, PCR success rates are calculated by locus and sample class (if a class file is provided). Locus specific PCR success rates are calculated based on an overall assessment of PCR success for each locus. Mathematically, this is the same as (1) above and calculates the number of successful amplification / the total number of amplifications attempted for each locus. Note, that locus specific PCR success rates are not calculated at the individual sample level. If a class file is provided, both the overall PCR success rate (1, above) and individual sample PCR success rate (2, above) are calculated for each sample class independently and overall. For example, if a class file identifies samples as being from Region A or Region B, the results will include separate overall PCR success and individual sample level PCR success rates for Region

A and Region B, as well as overall and individual sample level PCR success rates for the regions combined (i.e., both Region A and Region B).

*Genotyping Error Rates*.—Genotyping error is defined as the difference between the observed genotype (e.g., from a single replicate of a sample) and the true genotype, where we assume that the consensus genotype, or a reference genotype, reflects the true genotype. Quantifying genotyping errors is accomplished for each sample and locus by comparing each replicated genotype to the consensus genotype or reference genotype (Lampa et al. 2013). To quantify genotyping errors, we follow procedures detailed by Broquet and Petit (2004). Specifically, a FA is recorded when an allele not present in the consensus (or reference) genotype is observed in a single replicate. ADO is recorded when an allele present in the consensus (or reference) genotype is not observed in a single replicate. Methods employed by CONGENR differ slightly from that presented by Broquet and Petit (2004); specifically, Broquet and Petit (2004) suggest that ADO can only be documented for heterozygous loci, as ADO of a homozygous locus would result in a failure. Thus, the presence of a single allele that does not match a confirmed homozygous consensus (or reference) genotype would be scored as a FA, but not as ADO. CONGENR differs from this convention, by scoring this situation as both a FA and ADO. We have adopted this convention for two reasons. First, the presence of the FA indicates a successful PCR amplification, and failure of the confirmed allele (from the consensus or reference genotype) to amplify is by definition ADO. Second, the resulting output may be used as a diagnostic to identify samples for which the sizing standard in GENEMAPPER may be incorrectly labeled. In particular, the presence of both a FA and ADO within the same sample may reflect a shift (e.g. 2 base pair shift; in both homozygous and heterozygous genotypes), which may be an artifact resulting from an incorrectly labeled size standard. Recognition of this pattern can help researchers target samples for which the sizing standard should be re-evaluated. Additionally, if desired, researchers can easily identify sample replicates containing both a FA and ADO and make adjustments.

Similar to calculations of PCR success, genotyping error rates are calculated and reported by locus and sample class (if a class file is provided). Locus specific genotyping error rates are calculated as before, but for each locus independently. If a class file is provided, genotyping error rates are calculated for each sample class independently and overall. For example, if a class file identifies samples as being from Region A or Region B, the results will report separate genotyping errors (FA and ADO) for Region A and Region B, as well as overall combined genotyping error rates (one each for FA and ADO).

*Results Notation*.—When running the CONGENR in its entirety (i.e., from the `ConGen()` wrapper function), it is best practice to save the resulting R results list as an R object (as opposed to simply printing the results to the screen). An example of this is available below in the example for the `ConGen()` wrapper function. The `names(`*`object.name`*`)` function can then be used to view the names of the items contained within the resulting list object and either the $ or [[ ]] notation can be used to view the desired results. For example, to view the 'Rates' results, either `object.name$Rates` or `object.name[[10]]` will then return the desired results. Detailed descriptions of result files are provided below with each corresponding function, but in general, files ending in 'long' refer to genetic data formatted where each row represents one replicate-locus (for replicate data) or sample-locus (for consensus data) and combination. Files

ending in 'flat' refer to genetic data formatted where each row represents one replicate (for replicate data) or one sample (for consensus data) and each locus is represented by four columns (allowing for situation where >2 alleles are scored). For consensus genotype files, 0 denotes uncertainty, while NA represents a missing value (no successful amplification). For replicate genotype files, missing values are coded as either 0 or NA; files which code missing values as NA have names ending with 'na' (e.g., genotypes.complete.longna). The genotyping error results are returned in the 'long' format and for each sample-locus combination, contain a row for the consensus genotype (denoted by a '.con' added to the sample name) and one row for each replicate. Within the genotyping error results, FAs and ADOs are scored only for replicates (ADO and FA cannot be documented for consensus genotypes) and are coded as a binary response (1 = present, 0 = absent; no value indicates that genotyping errors could not be assessed based on number of replicates). Similarly, amplifications attempted (AA) and successful amplifications (SA) are tracked for each replicate and summarized for each consensus genotype as the total number of amplifications attempted (N.AA) and the number of successful amplifications (N.SA).

*Matching*.—Following the determination of consensus genotypes, it is often desirable to identify samples with identical (full match) or similar (partial match) multilocus genotypes. The `congen.matching()` function can be used to identify samples with fully or partially matching multilocus genotypes. Users can define the number of matching loci required to report two samples as being a match; in practice, the number of matches required will often be set to the number of loci required to meet desired levels of probability of identity (Waits et al. 2001). An option to consider or ignore loci with uncertainty (coded as an allele size = 0) provides a flexible framework for identifying matches even when uncertainty exists. When uncertainty is ignored, two samples are considered a match if the number of loci that are full matches and the number of loci with uncertainty sum to a value greater than the number of matches required. We caution that if datasets contain samples with substantial amounts of uncertainty (i.e., lower quality samples), results from matching procedures that ignore uncertainty can be extremely long, as samples with high levels of uncertainty will appear as a match to many (or all) of the other samples. Consequently, we recommend considering uncertainty when conducting initial matching analyses (note: the number of matches required can be decreased to increase matching tolerance). After close matches are determined, analyzing the consolidated dataset while ignoring uncertainty can produce more manageable results. When comparing samples, information on the spatial location at which each sample was collected can be beneficial. Matching methods employed by CONGENR allow the user to provide sample location data, which can be numeric (i.e., XY data, such as UTMs) or categorical (e.g., region, county, study area). Sample locations are provided in a separate file or data frame. When numeric locations are provided, the results will include a distance between each focal sample (the sample to which other samples are being compared) and each sample determined to be a match. Alternatively, if locations are categorical, the location of each sample will be added to the result file, facilitating comparisons between the focal sample and matching samples. By default, CONGENR attempts to minimize repetition in results generated. Consequently, if a sample is determined to be a perfect match to the focal sample, it is later skipped and not considered as a focal sample (i.e., if it is a perfect match to a sample already considered as the focal sample, it will produce the same results and therefore does not need to be analyzed further). Additionally, an identical set of samples that are not perfect matches are only reported once in the result file. For example, suppose both Sample B

and Sample C match the focal sample, Sample A, at all but 1 locus and no other samples are determined to be matches (based on the criteria set). When Sample B is considered as the focal sample, if *only* Sample A and Sample C are determined to be matches (based on the criteria set), then the results are omitted from the output, as they essentially report the same results as when considering Sample A as the focal sample (which would have already been included in the results file.

*Matching Results Notation*.—Matching with CONGENR will return a list and it is therefore best practice to save the resulting R results list as an R object. An example of this is available below in the example for the `congen.matching()` function. The `names(`*`object.name`*`)` function can then be used to view the names of the items contained within the resulting list object and either the $ or [[ ]] notation can be used to view the desired results. Detailed descriptions of results contained within the list are provided below. As with the consensus genotype files, 0 denotes uncertainty, while NA represents a missing value (no successful amplification). For replicate genotype files, missing values are coded as either 0 or NA. Matching results include a column for 'mismatches'. Focal samples (the sample to which subsequent samples are compared to) are denoted with a 'mismatches' value of 9999. Each subsequent 'mismatches' value represents the number of loci that do not match between the sample under consideration and the focal sample (when viewing results in the list format; via *`results.object$matching.list`*). Alternatively, when viewing results in the data frame format (via *`results.object$matching.data`*) each 'mismatches' value represents the number of loci that do not match between the sample under consideration and the preceding focal sample (denoted with 'mismatches' value 9999). All samples with zero matches (based on the criteria set) are located at the end of the results files; these samples are located in the last list element in the list formatted data, and the final rows in the data frame formatted results. When location data are provided, the 'Location' column is added to the results list and data frame. For categorical locations the location is reported for each sample (NA indicates that no location was available). For numeric locations, the Euclidean distance between the sample under consideration and the focal sample is reported (NA values indicate that the location was unavailable for either the location under consideration or the focal sample.

*Troubleshooting*.—The most common problems encountered relate to formatting of the input data. Additional columns of data, which are not required by CONGENR, may be included in input file, but will be ignored. The most common problem is the inclusion on non-numeric allele calls. For example, some researchers may record and X and Y for sex identification markers, as opposed to the numeric representation of their base pair lengths. As a quick diagnostic, once loaded into R, the input file data structure can be viewed using the `str()` function and each of the columns related to alleles should be numeric. Another problem which may be encountered results from data not having the same number of columns for every row. This typically results from files that were formatted using a text editor (e.g., Notepad) and where the trailing spaces were removed or deleted. This problem can be easily resolved by opening the file with software designed to handle spreadsheets, such as Microsoft Excel, ensuring columns are correctly aligned, and re-saving the file as tab delimited text file.

**Topics and Functions Documented:**

## Loading the CONGENR script into your R workspace

CONGENR is an R script and requires the user to first install the R programing language (available at www.r-project.org). The CONGENR script can be downloaded from http://www.uidaho.edu/cnr/research-outreach/facilities/leecg/publications-and-software and subsequently loaded into your R workspace through one of two processes.

Option 1: You can load the CONGENR script to your R console workspace by saving the downloaded CONGENR folder to your computer, extracting the compressed files, and then using the `source()` function to load the ConGen.r script file. `source()` simply requires the user to provide the appropriate directory path to the script file as a character string. After running the `source()` command with the appropriate file path, running `load.ConGenR()` is required to load necessary supporting packages.

For example, if the CONGENR folder was saved to *user1*'s desktop, the directory path and subsequent loading function may resemble the following:

```
> source("/Users/user1/Desktop/ConGenR/ConGen.r")
> load.ConGenR()
```

Similarly, if the CONGENR folder was saved to *user1*'s active working directory (the working directory active in the current session of the R console), '~' can be used to denote the working directory in the following manner:

```
> source("~/ConGenR/ConGen.r")
> load.ConGenR()
```

Note: you can determine the location of your working directory with the following function:

```
> getwd()
```

Option 2: You can open the ConGen.r script file, copy and paste the entire script into the R console, and then run the following code in the R console:

```
> load.ConGenR()
```

Example data files

**Description**

`ConGen` includes four example files that can be used to run examples included throughout this document. Example files listed below will need to be loaded to the users working directory manually to execute `ConGen` examples.

**Files**

| | |
|---|---|
| `ConGenR_example_data_long` | A .txt file containing 'long' format replicate data as exported from the program GENEMAPPER's genotypes table. This file conforms to the required input to run `ConGen`. |
| `ConGenR_example_data_class` | A .txt file containing the class information for each sample. The file contains 2 columns (Sample ID and class [called 'Freshness'] in this file). |
| `ConGenR_example_data_flat_replicates` | A .txt file containing 'flat' format replicate data, where each row represents one replicate and each locus has four columns (allowing for situation where >2 alleles are scored). Note: missing values for alleles (i.e., no allele scored) are denoted with a 0. |
| `ConGenR_example_data_flat_reference` | A .txt file containing 'flat' format reference data, where each row represents one sample and each locus has four columns (allowing for situation where >2 alleles are scored). Note: missing values for alleles (i.e., no allele scored) are denoted with NA; 0 indicates uncertainty. Additionally, by convention, Class is set to 0 for all reference or consensus genotypes. |

| | |
|---|---|
| `ConGenR_example_location_cat` | A .txt file containing categorical location data. Each row represents one sample and the file contains two columns. Column 1 contains the sample IDs and column 2 contains the location of the sample. Missing values, if included, would have been coded as NA. |
| `ConGenR_example_location_num` | A .txt file containing numeric (coordinate) location data. Each row represents one sample and the file contains three columns. Column 1 contains the sample IDs, while columns 2 and 3 contain the X and Y coordinate locations of the sample, respectively. Missing values are coded as NA. |

---

Required and supporting input files

---

**Required files**

Running CONGENR to determine consensus genotypes requires, at minimum, a file that contains one row per sample replicate–locus combination (i.e., a long format file) and the following columns: 'Sample File', 'Sample Name', 'Marker', 'Allele 1', 'Allele 2', 'Allele 3', and 'Allele 4'. Additional columns may be included in the input file, but will not be used. This file format can easily be exported from the program GENEMAPER by exporting the "genotypes table" with appropriate settings. An example of the data format, 'ConGenR_example_data_long', is provided in the CONGENR source folder.

Sample file and sample names cannot include a dash (i.e., "-"). If results in include dashes in the sample file or sample names, these should be replaced before running data. This can be easily accomplished in a text editing program (e.g., Notepad, Excel) using the replace function.

Note: If a long format file is unavailable, a flat format file (i.e., a file containing one row per replicate with columns for each locus and allele) can be converted to the required long format using the `flat.to.long()` function with the argument `reference = FALSE`, indicating that each row in the file contains replicate, as opposed to reference or consensus, genotypes. An example file, 'ConGenR_example_data_flat_replicates', which can be converted to the long format is provided in the CONGENR source folder.

**Supporting files**

CONGENR can take a 'Class' file, which assigns each sample to a specific 'class' and allows for the calculation and comparison of PCR success rates and genotyping error rates by sample classes. Sample class will typically constitute identifiable subdivisions that the research feels may influence PCR success and genotyping error rates. For example, classes may represent sample age, sample condition, sampling region, sampling season, sex (e.g. male vs. female), or age classes (e.g., juvenile vs adult). Any number of classes may be included.

When a classification file is included, the file should contain only 2 columns representing the Sample ID (column 1) and Class (column 2) of each sample. The Sample IDs contained in column 1 are case sensitive and should match the Sample IDs contained in the required long format data file. Although classes are categorical in nature, these classes should be represented in the input file as positive integers, excluding 999 (which is used to denote samples with unknown classification). Zeros should also not be used, as zeros are used internally to represent consensus or reference genotypes. If classes are known for some sample and not others (e.g., male, female, or unknown), input files should denote unknown samples as NA (missing values). If a 'Class' file is used, samples not represented in the 'Class' file will be omitted from the analysis. The 'Class' file may contain information for more samples than are included in the required long format file; this extra data will simply be ignored. An example file, 'ConGenR_example_data_class', is provided in the CONGENR source folder.

## Running CONGENR

Most functions within CONGENR can be run by calling a single function (i.e., `ConGen`), which calls necessary supporting functions. Alternatively, users may call supporting functions directly; this may be particularly useful if users wish to only calculate genotyping errors or convert files to alternative formats. Information on required inputs is provided with each function description below. Functions to execute matching procedures must be called directly.

### `load.ConGenR` Function to load CONGENR script and supporting packages

**Description**

`load.ConGenR` loads functions included in the CONGENR script into the users R working directory. Additionally, `load.ConGenR` checks the users working directory for the required supporting R package, 'stringr', and if it is not present, installs the package before loading it into the users R workspace.

**Usage**

```
load.ConGenR()
```

**Arguments**

No arguments are required.

| | |
|---|---|
| `ConGen` | Wrapper function that calls supporting functions to run a full analysis, including determining consensus genotypes |

**Description**

`ConGen` takes a long format data file (e.g., as produced by the program GENEMAPPER genotypes table) and compares replicates to establish consensus genotypes. `ConGen` then calculates PCR success rates (overall, individual sample level; each by locus) and if requested compares each replicate to the consensus genotype to determine genotyping error rates. If a class file is provided, `ConGen` calculates these metrics both overall and by class. Individual sample level PCR success rates are based on either 50% of the loci (when `PID = NULL`) or a user defined number of loci (typically set at the `PID`). `ConGen` allows the user to identify sex identification markers if they are present in the file and determine whether or not these markers should be included in the calculation of PCR and genotyping error rates. `ConGen` can be provided a list of sample IDs (e.g., samples that have been determined to be low quality or mixed) or prefixes (e.g., researchers may wish to exclude PCR negatives and may label these with the same prefix) which should be excluded from analysis; this option serves primarily to reduce computation time on samples of little or no interest.

**Usage**

```
ConGen(allele.calls, Class = NULL, rm.prefix = NULL, rm.vector
   = NULL, gte = TRUE, sex.id.markers = NULL, rm.sx.id = TRUE,
   PID = NULL)
```

**Arguments**

| | |
|---|---|
| `allele.calls` | A `data.frame` representing a GENEMAPPER genotypes table output file. This file can be generated in GENEMAPPER by exporting the genotypes table. Data is in the 'long' format, in which each row represents a single replicate-marker combination. Required columns include (at minimum): Sample.File, Sample.Name, Marker, Allele.1, Allele.2, Allele.3, and Allele.4. Even if <4 alleles are detected, the resulting columns should be present. |
| `Class` | A `data.frame` containing two columns representing Sample.ID and some numeric classification. `Class = NULL` (Default) indicates that no classifications are present. When Classes are present, classes should be represented by positive integers, excluding 999 (which is used to denote samples with unknown classification). Zero should not be used (0 is reserved for consensus genotypes). If classes are known for some sample and not others, input files should denote unknown samples as NA (missing values). |

| | |
|---|---|
| `rm.prefix` | A `vector` indicating character prefixes which identify sample to be removed from the analysis. For example, if you name PCR positives and negatives with the prefix 'PCR' and you would like to exclude these samples from the analyses, you can set `rm.prefix = c("PCR")`. Alternatively, `rm.prefix = NULL` (default) indicates that no samples should be removed based on this criteria. |
| `rm.vector` | A `vector` providing the names of samples that you would like to exclude from the analysis. For example, if you wish to not analyze samples 1001 and 1010, you can set `rm.vector = c(1001, 1010)`. Alternatively, `rm.vector = NULL` (default) indicates that no samples should be removed based on this criteria. |
| `gte` | A logical argument indicating whether or not genotyping error rates should be calculated. |
| `sex.id.markers` | A `vector` providing the names of sex identification markers. `sex.id.markers = NULL` (default) indicates that sex ID markers are not included. Alternatively, provide a `vector` with the names of the sex ID markers included in the dataset/analysis. Note, names are case sensitive and should match the marker names found in the `allele.calls`. |
| `rm.sx.id` | A logical argument indicating whether or not sex ID markers should be removed (excluded) when calculating individual sample level statistics. |
| `PID` | An integer value >0 and ≤ the number of loci (in most cases, this should be the number of loci required to meet the probability of identity requirements). Individual sample level success is then the proportion of samples with successful amplifications at ≥`PID` loci. Alternatively, when `PID = NULL`, individual sample level success will be the proportion of samples with successful amplifications at ≥50% of the loci. |

**Values**

| | |
|---|---|
| `samples.removed` | A `data.frame` containing the data for any samples removed through the `rm.prefix` or `rm.vector` arguments. |
| `duplicates.removed` | A `data.frame` representing duplicates within the `Class` file that were removed. |

| | |
|---|---|
| `genotypes.complete.long` | A `data.frame` containing the 'long' format data for all replicates analyzed and following the removal of unnecessary columns. Missing values for alleles are represented with 0. |
| `genotypes.complete.longna` | A `data.frame` containing the 'long' format data for all replicates analyzed and following the removal of unnecessary columns. Missing values for alleles are represented with NA. |
| `genotypes.complete.flat` | A `data.frame` containing the 'flat' format data for all replicates analyzed. Missing values for alleles are represented with 0. |
| `genotypes.complete.flatna` | A `data.frame` containing the 'flat' format data for all replicates analyzed. Missing values for alleles are represented with NA. |
| `consensus.genotypes.long` | A `data.frame` containing the 'long' format consensus genotypes resulting from the analyzed replicates. Missing values for are represented with NA; 0 is used to denote uncertainty. |
| `consensus.genotypes.flat` | A `data.frame` containing the 'flat' format consensus genotypes resulting from the analyzed replicates. Missing values for are represented with NA; 0 is used to denote uncertainty. |
| `genotyping.errors` | A `data.frame` containing the 'long' format consensus genotypes and replicates. For allele calls, missing values within each consensus genotype are denoted with NA, while 0 indicates uncertainty. Rows representing replicates include 0 for missing values. For each replicate row, the presence of a FA and ADO, as well as successful amplifications (SA), are coded as a binary response (1 = present, 0 = absent). For each consensus genotype row, the number of amplifications attempted (AA) and SA are recorded. |
| `Rates` | A `data.frame` representing overall PCR success, individual sample level PCR success, genotyping error rates (FA and ADO) and the number of unique samples analyzed, both overall and by class (when included). |

| | |
|---|---|
| `Locus.Rates` | A `data.frame` representing overall PCR success and genotyping error rates (FA and ADO) by locus. |
| `Success.by.NLoci` | A `data.frame` summarizing the number of samples that achieved a consensus genotype at 0 to *n* loci, where *n* is the number of loci included. If `rm.sx.id = TRUE`, sex ID markers are excluded. |
| `Success.by.Sample` | A `data.frame` identifying the number of loci for which a consensus genotype was achieved for each sample. If `rm.sx.id = TRUE`, sex ID markers are excluded. |

**Example**

```
#After loading the example data files:
#Example: Includes a class file, removes samples starting
#with 'PCR', removes the sample 'DPG1', calculates genotyping
#error rates, removes sex ID markers from individual sample
#level statistics, and determines individual sample level
#statistics based on probability of identity = 7 loci.


> example.results <- ConGen(allele.calls =
        ConGenR_example_data_long, Class =
        ConGenR_example.data.class, rm.prefix = c("PCR"),
        rm.vector = c("DPG1"), gte = TRUE, sex.id.markers =
        c("SexIDPrimer-X", "SexIDPrimer-Y"), rm.sx.id = TRUE,
        PID = 7)

#View the names of the values in the resulting example file
> names(example.results)

#View the Rates calculated
> example.results$Rates
```

| | |
|---|---|
| `gt.error` | Function that calculates genotyping error rates, including false alleles (FA) and allelic dropout (ADO) |

**Description**

`gt.error` takes two input files including a `reference` file and `replicates` file. The `gt.error` function is called directly by the `ConGen` function, which passes the necessary files to it, but may be run independently. Both the `reference` file and `replicates` file should be in the 'long' format (i.e., one row per sample-locus combination or replicate-locus combination, respectively). Additional details for each input file are listed below under the Arguments. The `gt.error` function compares each replicate to the consensus, or reference, genotype to determine the presence of FAs and/or ADO, and summarizes genotyping error rates overall, by sample class (if provided), and by locus.

**Usage**

```
gt.error(reference, replicates)
```

**Arguments**

| | |
|---|---|
| `reference` | A `data.frame` containing one row for each sample-locus combination. Genotypes should represent either the consensus genotypes (i.e., determined through replicated samples) or reference genotypes (generated from a high quality samples). Data must contain at least 10 columns that align with the first 10 columns of the `replicates` file (details below). Any columns beyond the first 10 will not be used and will be discarded. Note: Numeric classes for all samples in the `reference` file should be the same; all should be set to 0. Additionally, for each Allele.i column (see below), the numeric allele size should be recorded. When no amplification is observed, NA should be used (this differs from the `replicates` file). |
| `replicates` | A `data.frame` containing one row for each replicate-locus combination. Data must contain at least 10 columns that align with the first 10 columns of the `reference` file (details below). Note: Numeric classes for all samples in the `replicates` file should be positive integers (excluding 999). Additionally, for each Allele.i column, the numeric allele size should be recorded. When no amplification is observed, 0 should be used (this differs from the `reference` file). |

The first 10 columns of each file should include the following (in this order):

1. Sample.ID (factor) - should be the same across replicates of the same sample.
2. Class (numeric) - positive integer (see notes for differences between `reference` file and `replicates` file).
3. Sample.File (factor) - often contains Sample.ID and some descriptor (value) to distinguish among replicates.
4. Sample.Name (factor) - often the same or similar to the Sample.ID, but may contain additional information or descriptors.
5. Marker (factor) - contains the name of the marker for which the data is associated.
6. Allele.1 (numeric) - Numeric value for the size in base-pairs of the allele.
7. Allele.2 (numeric) - Numeric value for the size in base-pairs of the allele.
8. Allele.3 (numeric) - Numeric value for the size in base-pairs of the allele.
9. Allele.4 (numeric) - Numeric value for the size in base-pairs of the allele.
10. GenoType (factor) - there are six possible levels including ">2_Alleles", "Heterozygote", "Homozygote", "Sample_Failed", "Homozygote_wUnc", "Incomplete".

**Values**

`genotyping.errors` — A `data.frame` containing the 'long' format consensus genotypes and replicate data. For allele calls, missing values within each consensus genotype are denoted with NA, while 0 indicates uncertainty. Rows representing replicates include 0 for missing values. For each replicate row, the presence of a FA and ADO, as well as successful amplifications (SA), are coded as a binary response (1 = present, 0 = absent). For each consensus genotype row, the number of amplifications attempted (AA) and SA are recorded.

`Class.Rates` — A `data.frame` representing PCR success and genotyping error rates (FA and ADO) by class (when included).

`Locus.Rates` — A `data.frame` representing PCR success and genotyping error rates (FA and ADO) by locus.

| | |
|---|---|
| `Overall.PCR.Success` | The overall PCR success rate across all samples and loci. |
| `Overall.ADO` | The overall ADO rate across all samples and loci. |
| `Overall.FA` | The overall FA rate across all samples and loci. |

**Example**

```
#After loading the example data files:
#Example: Identify the names of the loci in the file (i.e.,
#ex.loci), convert the flat format example files to the
#required long format with the flat.to.long function and then
#calculate the genotyping error rates.

> ex.loci <- c("Locus1", "Locus2", "Locus3", "Locus4",
        "Locus5", "Locus6", "Locus7", "Locus8", "Locus9",
        "SexIDPrimer.X", "SexIDPrimer.Y")

> example.data.ref.long <- flat.to.long(gen.data.flat =
        ConGenR_example_data_flat_reference, loci.vec =
        ex.loci, reference = TRUE)

> example.data.rep.long <- flat.to.long(gen.data.flat =
        ConGenR_example_data_flat_replicates, loci.vec =
        ex.loci, reference=FALSE)

> example.results <- gt.error(reference=example.data.ref.long,
        replicates = example.data.rep.long)

#View the names of the values in the resulting example file
> names(example.results)

#View the FA and ADO Rates by locus
> example.results$Locus.Rates
```

| pcr.success | Function that calculates overall PCR success rates |
|---|---|

**Description**

pcr.success is called by either the ConGen or gt.error functions and calculates the overall PCR success rate as the number of successful amplification / the total number of amplifications attempted. The gen.data file should be in the 'long' format (i.e., one row per replicate-locus combination, respectively) and contain at minimum the Additional details for each input file are listed below under the Arguments. The gt.error function compares each replicate to the consensus, or reference, genotype to determine the presence of FAs and/or ADO, and summarizes genotyping error rates overall, by sample class (if provided), and by locus.

**Usage**

```
pcr.success(gen.data)
```

**Arguments**

| | |
|---|---|
| gen.data | A data.frame in the 'long format containing one row for each sample-locus combination. Missing values for alleles should be denoted with a 0 or NA. |

**Values**

| | |
|---|---|
| Overall.PCR.Success | The overall PCR success rate across all samples and loci. |
| Class.PCR.Success | A named vector representing PCR success rates by class (when included). |
| Locus.PCR.Success | A named vector representing PCR success by locus. |

**Example**

```
#After loading the example data files:
#Example: In order to determine success by class, the class
#must be added to the ConGenR_example_data_long file. When
#pcr.success is called directly from the ConGen function, it
#handles this internally. If calling the pcr.success function
#independently, the add.class function must first be used to
#ensure the data is properly formatted (i.e., includes a
#'Class' column). If 'Class' is not set to NULL, a Sample.ID
#column is required.


> example.data <- cbind(Sample.ID =
        ConGenR_example_data_long$Sample.Name,
        ConGenR_example_data_long)

> example.data <- add.class(Class =
        ConGenR_example_data_class, gen.data =
        example.data)[[1]]

> example.results <- pcr.success(gen.data = example.data)

#View the resulting PCR success rates
> example.results
```

| sample.pcr.success | Function that calculates individual sample level PCR success |
|---|---|

**Description**

sample.pcr.success is called by the ConGen function and calculates the individual sample level PCR success rate, or the proportion of samples that had successful amplifications at *n* loci. When the argument PID = NULL, *n* = 50% of the loci. Otherwise, PID may be set to any value between 1 and the total number of loci and *n* = PID. In practice, PID will most appropriately be set to the probability of identity (Waits et al. 2001) required for successful identification of individuals.

**Usage**

```
sample.pcr.success(gen.data.long, gen.data.flat, PID = NULL,
   sex.id.markers = NULL, rm.sx.id = TRUE)
```

**Arguments**

| | |
|---|---|
| gen.data.long | A data.frame in the 'long' format containing one row for each sample-locus combination. Missing values for alleles should be denoted with a 0. |
| gen.data.flat | A data.frame containing the 'flat' format data for all replicates analyzed. Missing values for alleles should be denoted with a 0. |
| PID | An integer value >0 and ≤ the number of loci (in most cases, this should be the number of loci required to meet the probability of identity requirements). Individual sample level success is the proportion of samples with successful amplifications at ≥PID loci. Alternatively, when PID = NULL, individual sample level success is the proportion of samples with successful amplifications at ≥50% of the loci. |
| sex.id.markers | A vector providing the names of sex identification markers. sex.id.markers = NULL (default) indicates that sex ID markers are not included. Alternatively, provide a vector with the names of the sex ID markers included in the dataset/analysis. |
| rm.sx.id | A logical argument indicating whether or not sex ID markers should be removed when calculating sample level statistics. |

**Values**

| | |
|---|---|
| Overall.Sample.PCR.Success | The overall individual sample level PCR success rate across all samples and considering *n* = PID loci. |

| | |
|---|---|
| `Sample.PCR.Success` | A `named vector` representing individual sample level PCR success rates by class (when included) and considering *n* = `PID` loci. |
| `Unique.Samples` | A `named vector` representing the number of samples within each class. |

**Example**

```
#After loading the example data files:
#Example: In order to determine success by class, the class
#must be added to the ConGenR_example_data_long file. When
#sample.pcr.success is called directly from the ConGen
#function, it handles this internally. If calling the
#sample.pcr.success function independently, the add.class
#function must first be used to ensure the data is properly
#formatted (i.e., includes a 'Class' column). If 'Class' is
#not set to NULL, a Sample.ID column is required.

> example.data <- cbind(Sample.ID =
        ConGenR_example_data_long$Sample.Name,
        ConGenR_example_data_long)

> example.data <- add.class(Class =
        ConGenR_example_data_class, gen.data =
        example.data)[[1]]

> example.results <- sample.pcr.success(gen.data.long =
        example.data, gen.data.flat =
        ConGenR_example_data_flat_replicates, PID = NULL,
        sex.id.markers = NULL, rm.sx.id = TRUE)

#View the resulting individual sample level PCR success rates
> example.results
```

| rm.samples | Function that removes identified samples from the analysis |
|---|---|

**Description**

rm.samples is called by the ConGen function and removes those samples identified by either the rm.prefix or rm.vector arguments.

**Usage**

```
rm.samples(rm.prefix = NULL, rm.vector = NULL, gen.data)
```

**Arguments**

| | |
|---|---|
| rm.prefix | A vector indicating character prefixes which identify sample to be removed from the analysis. This prefix is passed to rm.samples by the ConGen function. |
| rm.vector | A vector providing the names of samples that you would like to exclude from the analysis. This vector of sample names is passed to rm.samples by the ConGen function. |
| gen.data | A data.frame containing genetic data in the 'long' format, where each row represents a single replicate-marker combination. This file is passed to rm.samples by the ConGen function. |

**Values**

| | |
|---|---|
| allele.calls | A new data.frame containing genetic data in the 'long' format, excluding those samples that were removed. |
| samples.removed | A data.frame containing information on the samples removed. |

| | |
|---|---|
| add.class | Function that combines the supporting class information to the genetic data. |

**Description**

add.class is called by the ConGen function and merges the Class file with the genetic data. Samples not included in the class file are excluded, and therefore all samples that are to be analyzed should have a class assigned (Note: if only some samples have a known class, 999 should be used by convention to represent the class of unknown samples). add.class may be useful for formatting files not processed by the ConGen function, as can be seen in the example for the sample.pcr.success function.

**Usage**

```
add.class(Class=NULL, gen.data)
```

**Arguments**

| | |
|---|---|
| Class | A data.frame containing two columns representing Sample.ID and some numeric classification. Class = NULL (Default) indicates that no classifications are present. When Classes are present, classes should be represented by positive integers, excluding 999 (which is used to denote samples with unknown classification). Zero should not be used (0 is reserved for consensus genotypes). If classes are known for some sample and not others, input files should denote unknown samples as NA (missing values). |
| gen.data | A data.frame containing genetic data in the 'long' format, where each row represents a single replicate-marker combination. This file is passed to add.samples by the ConGen function. |

**Values**

| | |
|---|---|
| allele.calls | A new data.frame containing genetic data in the 'long' format, excluding those samples that were removed. |
| duplicates.removed | A data.frame containing information on any samples represented more than once in the Class file. |

| | |
|---|---|
| `det.congen` | Function that compares replicated genetic samples and determines the consensus genotype. |

**Description**

`det.congen` is called by the `ConGen` function and employs common protocols for determining consensus genotypes from replicated (muli-tubes approach) DNA samples (e.g., Frantz et al. 2003, Flagstad et al. 2004, Zhan et al. 2010). `det.congen` requires that single-locus genotypes be observed at least twice for heterozygous and three times for homozygous genotypes. Consensus genotypes may include uncertainty, which are characterized by either the confirmation of one allele (i.e., observed 2 – 4 times) while a second allele is observed only once, or the observation of a single allele only once.

**Usage**

```
det.congen(alleles)
```

**Arguments**

| | |
|---|---|
| `alleles` | A `data.frame` containing genetic data in the 'long' format, where each row represents a single replicate-marker combination. This file is passed to `det.congen` by the `ConGen` function. |

**Values**

| | |
|---|---|
| `consensus.genotypes.long` | A `data.frame` containing the 'long' format consensus genotypes resulting from the analyzed replicates. Missing values for are represented with NA; 0 is used to denote uncertainty. |
| `consensus.genotypes.flat` | A `data.frame` containing the 'flat' format consensus genotypes resulting from the analyzed replicates. Missing values for are represented with NA; 0 is used to denote uncertainty. |

| | |
|---|---|
| `det.genotype` | Function that determines the type of genotype observed for a replicate. |

**Description**

`det.genotype` is called by the `ConGen` function and is an internal function that determines the type of genotype observed for each replicate. Specifically, `det.genotype` determines if the replicates genotype is heterozygous (i.e., 2 alleles observed) or homozygous (i.e., 1 allele observed), or if no alleles (i.e. sample failed) or >2 alleles were observed. This function facilitates other analyses and is not intended to provide the final determination of genotypes.

**Usage**

```
det.genotype(alleles)
```

**Arguments**

| | |
|---|---|
| `alleles` | A `data.frame` containing genetic data in the 'long' format, where each row represents a single replicate-marker combination. This file is passed to `det.genotype` by the `ConGen` function. |

**Values**

| | |
|---|---|
| `GenoType` | A `vector` containing the genotype determination (i.e., ‘Sample Failed’, ‘Homozygote’, ‘Heterozygote’, or ‘>2 alleles’) for each replicate. |

| | |
|---|---|
| samp.details | Function that returns a summaries of the number of loci for which a consensus genotype has been achieved. |

**Description**

samp.details is called by the ConGen function and is an internal function that summarizes and reports the number of loci for which a consensus genotype was achieved. samp.details considers only genotypes without uncertainty as consensus genotypes.

**Usage**

```
samp.details(gen.data.long, con.gen.flat, sex.id.markers =
   NULL, rm.sx.id = TRUE)
```

**Arguments**

| | |
|---|---|
| gen.data.long | A data.frame containing genetic data in the 'long' format, where each row represents a single replicate-marker combination. This file is passed to samp.details by the ConGen function. |
| con.gen.flat | A data.frame containing the 'flat' format consensus genotype data for all samples analyzed. Missing values for alleles should be denoted with a 0. This file is passed to samp.details by the ConGen function. |
| sex.id.markers | A vector providing the names of sex identification markers. sex.id.markers = NULL (default) indicates that sex ID markers are not included. Alternatively, provide a vector with the names of the sex ID markers included in the dataset/analysis. . |
| rm.sx.id | A logical argument indicating whether or not sex ID markers should be removed when calculating individual sample level statistics. |

.

**Values**

| | |
|---|---|
| Success.by.Number.of.Loci | A data.frame summarizing the number of samples that achieved a consensus genotype at 0 to *n* loci, where *n* is the number of loci included. If rm.sx.id = TRUE, sex ID markers are excluded. |
| Success.by.Sample | A data.frame identifying the number of loci for which a consensus genotype was achieved for each sample. If rm.sx.id = TRUE, sex ID markers are excluded. |

| | |
|---|---|
| `sort.alleles.long` | Function that re-sorts alleles within 'long' format files following analysis to so that alleles conform to standard orders. |

**Description**

`sort.alleles.long` is called by the `ConGen` function and re-sorts the alleles within a 'long' format file for each sample and locus so that the allele order conforms to the standard ordering. Specifically, `sort.alleles.long` ensures that smaller alleles of a heterozygous genotype appear first (e.g., 70/74 as opposed to 74/70) and that uncertainty in allele calls appear after confirmed alleles (e.g., 70/0 as opposed to 0/70) for genotypes with uncertainty. `sort.alleles.long` is intended to be an internal function that cleans data following analyses that often result in sorting of alleles into a non-standard order.

**Usage**

```
sort.alleles.long(z)
```

**Arguments**

| | |
|---|---|
| `z` | A `data.frame` containing genetic data in the 'long' format, where each row represents a single replicate-marker combination. This file is passed to `sort.alleles.long` by the `ConGen` function. |

**Values**

| | |
|---|---|
| `z` | A `data.frame` containing the 'long' format genotype data where the order of the alleles conforms to standard reporting. |

| | |
|---|---|
| `sort.alleles.flat` | Function that re-sorts alleles within 'flat' format files following analysis to so that alleles conform to standard orders. |

**Description**

`sort.alleles.flat` is called by the `ConGen` function and re-sorts the alleles within a 'flat' format file for each sample and locus combination so that the allele orders conform to standard ordering. Specifically, `sort.alleles.flat` ensures that smaller alleles of a heterozygous genotype appear first (e.g., 70/74 as opposed to 74/70) and that uncertainty in allele calls appear after confirmed alleles (e.g., 70/0 as opposed to 0/70) for genotypes with uncertainty. `sort.alleles.long` is intended to be an internal function that cleans data following analyses that often result in sorting of alleles into a non-standard order.

**Usage**

```
sort.alleles.flat(z)
```

**Arguments**

| | |
|---|---|
| `z` | A `data.frame` containing genetic data in the 'flat' format, where each row represents a single replicate-marker combination. This file is passed to `sort.alleles.flat` by the `ConGen` function. |

**Values**

| | |
|---|---|
| `z` | A `data.frame` containing the 'flat' format genotype data where the order of the alleles. |

`long.to.flat.format` Function that converts ‘long’ format data to the ‘flat’ format.

**Description**

`long.to.flat.format` is called by the `ConGen` function and reformats ‘long’ format data (i.e., data where each row represents a replicate-locus combination) into the ‘flat’ format (i.e., each row represents a replicate with loci represented by multiple columns).

**Usage**

```
long.to.flat.format(alleles)
```

**Arguments**

| | |
|---|---|
| `alleles` | A `data.frame` containing genetic data in the 'long' format, where each row represents a single replicate-marker combination. This file is passed to `long.to.flat.format` by the `ConGen` function. |

**Values**

| | |
|---|---|
| `genotypes.complete.flat` | A new `data.frame` containing genetic data in the 'flat' format. Missing values for alleles are denoted by 0. |
| `genotypes.complete.flatna` | A new `data.frame` containing genetic data in the 'flat' format. Missing values for alleles are denoted by NA. |

| | |
|---|---|
| `flat.to.long` | Function that converts ‘flat’ format data to the ‘long’ format. |

**Description**

`flat.to.long` reformats ‘flat’ formatted data (i.e., each row represents either a replicate [for replicate data ] or a sample [for consensus or reference genotypes] with loci represented by multiple columns) into ‘long’ formatted (i.e., data where each row represents a replicate-locus or sample-locus combination, respectively). This function is not called by the `ConGen` function, but assists users with getting data in the ‘flat’ format reformatted for use with the `ConGen` or `gt.error` functions.

**Usage**

```
flat.to.long(gen.data.flat, loci.vec, reference = FALSE)
```

**Arguments**

| | |
|---|---|
| `gen.data.flat` | A `data.frame` containing genetic data in the 'flat' format, where each row represents either a replicate (for replicate data) or a sample (for consensus or reference genotypes) with loci represented by multiple columns. |
| `loci.vec` | A `vector` containing the names of the loci contained within `gen.data.flat`. Names of loci are case sensitive and should match the representation of the names in `gen.data.flat`. Within `loci.vec` each names should be a character string. |
| `reference` | A logical argument indicating whether or not the input genetic data represents consensus genotypes (or reference genotypes). If `reference = FALSE` (Default), the input data represents replicate data. |

**Values**

| | |
|---|---|
| `long.format.gen.data` | A new `data.frame` containing genetic data in the 'long' format. |

**Example**

```
#After loading the example data files
#Example: Identify the names of the loci in the file (i.e.,
#ex.loci), convert the flat format example files to the
#required long format with the flat.to.long function and then
#calculate the genotyping error rates.

> ex.loci <- c("Locus1", "Locus2", "Locus3", "Locus4",
        "Locus5", "Locus6", "Locus7", "Locus8", "Locus9",
        "SexIDPrimer.X", "SexIDPrimer.Y")

> example.data.ref.long <- flat.to.long(gen.data.flat =
        ConGenR_example_data_flat_reference, loci.vec =
        ex.loci, reference = TRUE)

> example.data.rep.long <- flat.to.long(gen.data.flat =
        ConGenR_example_data_flat_replicates, loci.vec =
        ex.loci, reference=FALSE)
```

| | |
|---|---|
| `congen.matching` | Function that identifies samples that have fully or partially matching multilocus genotypes |

**Description**

`congen.matching` takes a flat format consensus genotype file (e.g., as produced by the `ConGen` function) and either a long format consensus genotype file (e.g., as produced by the `ConGen` function, from which to extract loci information) or a vector identifying the loci contained within the flat format file. The `congen.matching` function then compares the multilocus genotypes of samples to one another and provides a simple and condensed output of matches and partial matches (based on criteria set). If numeric location data are provided, `congen.matching` calculates the distance between the focal sample (the sample to which other sample are compared) and each full or partial match. Otherwise, if categorical location data are provided, the function appends the location data to the output to facilitate comparisons. If location data are provided, the location data type should be identified in the appropriate argument. By default, the `congen.matching` function returns full matches only, but this can be modified by setting the argument for matches required to a value less than the total number of loci. By default the `congen.matching` function only reports perfect matches once, but this can be overridden. The user may consider or ignore loci with uncertainty (i.e., alleles coded as 0 indicating uncertainty in the final consensus genotype). To consider uncertainty indicates that genotypes must match perfectly (e.g., 144/0 = 144/0 but 144/0 ≠ 144/144), while ignoring uncertainty includes comparisons with uncertainty as matches (e.g., 144/0 = 144/144). Like `ConGen`, `congen.matching` allows the user to identify sex identification markers if they are present in the file and determine whether or not these markers should be included in the matching procedures.

**Usage**

```
congen.matching(congen.data.flat, congen.data.long = NULL,
   loci.names = NULL, matches.required = NULL, sex.id.markers
   = NULL, rm.sx.id = TRUE, consider.uncertainty = TRUE,
   skip.matches = TRUE, location.data = NULL,
   location.data.type = NULL)
```

**Arguments**

| | |
|---|---|
| `congen.data.flat` | A `data.frame` representing consensus genotypes in the flat format. This file is generated as an output of the `ConGen` function. At minimum, this file should have a 'Sample.ID' column (first column), followed by ≥2 columns for each locus or marker under consideration. Each locus or marker column name must begin with the name of the locus (as identified by the `congen.data.long` file or the `loci.names` argument) and each locus or marker should be represented |

| | |
|---|---|
| | by ≥2 columns (only the first 2 columns will be considered and any additional columns will be ignored). |
| `congen.data.long` | A `data.frame` representing consensus genotypes in the long format. This file is generated as an output of the `ConGen` function. If the resulting 'consensus.genotypes.long' file is not available from the `ConGen` function, then provide the names of the loci to `loci.names` and leave `congen.data.long = NULL` (Default). |
| `loci.names` | A `vector` with the names of loci (or markers) included in the `congen.data.flat` file. `loci.names = NULL` (default) indicates that a `congen.data.long` file has been included, from which the name of the loci (or markers) will be extracted. Note, names are case sensitive and should match the marker names found in `congen.data.flat`. |
| `matches.required` | A numeric value between 1 and the number of loci under consideration. This value represents the number of loci (or markers) that must match in order for two sample to be considered a match. Often, this value will be selected to align with the number of loci required to meet desired levels of probability of identity. Alternatively, `matches.required = NULL` (default) indicates that samples must match at all markers under consideration to be considered a match. |
| `sex.id.markers` | A `vector` providing the names of sex identification markers. `sex.id.markers = NULL` (default) indicates that sex ID markers are not included. Alternatively, provide a `vector` with the names of the sex ID markers included in the dataset/analysis. Note, names are case sensitive and should match the marker names found in `congen.data.flat`. |
| `rm.sx.id` | A logical argument indicating whether or not sex ID markers should be removed (excluded) when conducting matching procedures. |
| `consider.uncertainty` | A logical argument indicating whether or not loci with uncertainty must be a perfect match to be considered a match. Setting `consider.uncertainty = FALSE` allows loci containing uncertainty to be ignored. |

| | |
|---|---|
| `skip.matches` | A logical argument indicating whether or not samples that are a perfect match to a sample already analyzed should be skipped. Setting `skip.matches = TRUE` (Default) will avoid analyzing samples that have already been matched to another sample as a full match (at all loci or markers) as the focal sample, will reduce computational time, and will shorten the results by reducing duplicate comparisons. |
| `location.data` | A `data.frame` containing location data for samples being matched. This file should contain 2 (for categorical locations) or 3 (for numeric locations) columns. The first column in both cases should contain the Sample IDs (these should match those in the `congen.data.flat` [case sensitive]). For categorical location data, the second column should include a categorical location index. For numeric location data, columns 2 and 3 should contain the X and Y location coordinates, respectively. The `location.data` file may include more samples than contained in the `congen.data.flat` file. If location data are not available for all samples in the `congen.data.flat` data, these samples should still be included with NA used to denote their locations. |
| `location.data.type` | A `character string` indicating whether or not the location data provided is 'Numeric' or 'Categorical'. `location.data.type = NULL` (Default) indicates that no location file is provided. |

**Values**

| | |
|---|---|
| `Matches.required` | A value indicating the number of matches required in the analysis. |
| `consider.uncertainty` | A logical response indicating the condition used in the analysis. |
| `skip.matches` | A logical response indicating the condition used in the analysis. |
| `matching.data` | A `data.frame` containing the results of the matching analysis. Focal samples (those samples to which other samples were compared) are denoted with a 'Mismatches' value of 9999. The 'Mismatches' value for samples being compared indicate the number of loci (or markers) that differed between the preceding focal sample and the |

| | |
|---|---|
| | sample under consideration. When location data are provided, categorical locations or the Euclidian distance between the sample under consideration and the preceding focal sample are appended as the last column. Samples with no matches are stored at the end of the file and have a 'Mismatches' value of 9999 (with no comparative samples after). This file contains the same information as the `matching.list` data, but has been collapsed into a single `data.frame`. |
| `matching.list` | A `list` containing the results of the matching analysis. Each list element contains a comparison between one focal sample (the sample to which other samples were compared; the top row in the list element) and all samples determined to be a match to the focal sample. The last list element, contains all of the samples that were determined to have no matches. Within each list element, 'Mismatches' = 9999 indicates the focal sample (row one except in the last list element) and all other 'Mismatches' values indicate the number of loci (or markers) that differed between the focal sample and the sample under consideration. Location data are included in the same way as with `matching.data`. |

**Example**

```
#After loading the example data files:
#Example: First, rerun the ConGen() example to create input
#files. Then, run the matching procedures with no location
#data, with numeric location data, and with categorical
#location data.


> example.results<-ConGen(allele.calls =
        ConGenR_example_data_long,
        Class = ConGenR_example_data_class,
        rm.prefix = c("PCR"), rm.vector = c("DPG1"),
        gte = TRUE, sex.id.markers = c("SexIDPrimer-X",
        "SexIDPrimer-Y") , rm.sx.id = TRUE, PID = 7)
```

```
> example.matching.noloc<-congen.matching(congen.data.flat =
      example.results$consensus.genotypes.flat,
      congen.data.long =
      example.results$consensus.genotypes.long, loci.names =
      NULL, matches.required = 9, sex.id.markers =
      c("SexIDPrimer-X", "SexIDPrimer-Y"), rm.sx.id = FALSE,
      consider.uncertainty = TRUE, skip.matches = TRUE,
      location.data = NULL, location.data.type = NULL)

> names(example.matching.noloc)
> example.matching.noloc$matching.list #View list results

> example.matching.numloc<-congen.matching(congen.data.flat
      = example.results$consensus.genotypes.flat,
      congen.data.long =
      example.results$consensus.genotypes.long, loci.names =
      NULL, matches.required = 9, sex.id.markers =
      c("SexIDPrimer-X", "SexIDPrimer-Y"), rm.sx.id = FALSE,
      consider.uncertainty = TRUE, skip.matches = TRUE,
      location.data = ConGenR_example_location_num,
      location.data.type = "Numeric")

> example.matching.numloc$matching.list

> example.matching.catloc<-congen.matching(congen.data.flat
      = example.results$consensus.genotypes.flat,
      congen.data.long =
      example.results$consensus.genotypes.long, loci.names =
      NULL, matches.required = 9, sex.id.markers =
      c("SexIDPrimer-X", "SexIDPrimer-Y"), rm.sx.id = FALSE,
      consider.uncertainty = TRUE, skip.matches = TRUE,
      location.data = ConGenR_example_location_cat,
      location.data.type = "Categorical")

> example.matching.catloc$matching.list
```

| | |
|---|---|
| add.loc | Function that combines the supporting location information to matching data. |

**Description**

add.loc is called by the congen.matching function and merges the location.data file with the results of the matching. All samples present in the matching results (i.e., in the flat formatted consensus genotypes used in the matching) should be present in the location.data file (though values may be set to NA if location data for some samples are unknown). When location data are numeric, add.loc calculates the Euclidean distance between matches and merges this value with the results of the matching.

**Usage**

```
add.class(list.data, location.data, location.data.type =
   c("Numeric", "Categorical"))
```

**Arguments**

| | |
|---|---|
| list.data | A list containing results from congen.matching. This argument is passed from the congen.matching function. |
| location.data | A data.frame containing location data for samples. This argument is passed from the congen.matching function. See the congen.matching function description for details on format. |
| location.data.type | A character string indicating whether or not the location data provided is 'Numeric' or 'Categorical'. This argument is passed from the congen.matching function. See the congen.matching function description for details on format. |

**Values**

| | |
|---|---|
| matching.list | A list containing the results of the matching analysis with location data appended. This list is returned internally to the congen.matching function. See the congen.matching function description for details on format. |

## Literature Cited

Broquet T, Petit E (2004) Quantifying genotyping errors in noninvasive population genetics. Mol Ecol 13:3601–3608.

Flagstad Ø, Hedmark E, Landa A, Brøseth H, Persson J, Andersen R, Segerström P, Ellegren H (2004) Colonization history and noninvasive monitoring of a reestablished wolverine population. Conserv Biol 18:676–688.

Frantz AC, Pope LC, Carpenter PJ, Roper TJ, Wilson GJ, Delahay RJ, Burke T (2003) Reliable microsatellite genotyping of the Eurasian badger (*Meles meles*) using faecal DNA. Mol Ecol 12:1649–1661.

Lampa S, Henle K, Klenke R, Hoehn M, Gruber B (2013) How to overcome genotyping errors in non-invasive genetic mark-recapture population size estimation–a review of available methods illustrated by a case study. J Wildl Manage 77:1490–1511.

R Core Team (2015). R: a language and environment for statistical computing. R Foundation for Statistical Computing, Vienna, Austria.

Taberlet P, Griffin S, Goossens B, Questiau S, Manceau V, Escaravage N, Waits LP, Bouvet J (1996) Reliable genotyping of samples with very low DNA quantities using PCR. Nucleic Acids Res 24:3189–3194.