

# Instruction of functions of R package *KINSIMU*

Guanju Ma, Shujin Li

January 8, 2024

## Contents

<b>1</b>	<b>Brief introduction</b>	<b>2</b>
<b>2</b>	<b>Install and library the <i>KINSIMU</i> package</b>	<b>2</b>
<b>3</b>	<b>Functions in the package</b>	<b>2</b>
3.1	' <i>EvaluatePanel()</i> ' . . . . .	2
3.2	' <i>pairsimu()</i> ' . . . . .	4
3.3	' <i>pedisimu()</i> ' . . . . .	6
3.4	' <i>LRparas()</i> ' . . . . .	7
3.5	' <i>trioPI()</i> ' . . . . .	10
3.6	' <i>IICAL()</i> ' . . . . .	12
3.7	' <i>LRhsip()</i> ' . . . . .	13
3.8	' <i>LRgpgcam()</i> ' . . . . .	15
3.9	' <i>logLR()</i> ' . . . . .	17
3.10	' <i>testsimulation()</i> ' . . . . .	20
3.11	' <i>outputCSV()</i> ' . . . . .	21
<b>4</b>	<b>Data involved</b>	<b>21</b>
4.1	<i>FortytwoSTR</i> . . . . .	21
4.2	<i>pediexample</i> . . . . .	22

# 1 Brief introduction

The R package *KINSIMU* has been developed using the unified formulae for the assessment and simulation of specific panels in kinship analysis, we hereby present instructions of several functions in the package.

## 2 Install and library the *KINSIMU* package

The authors have uploaded the package to CRAN, making it able to be install directly with the following code:

```
1 install.packages("KINSIMU")
```

Otherwise, the repository file "KINSIMU.0.1.1.tar.gz" is provided as File S2 of this study, and the package can be installed using the subsequent code after downloading it:

```
1 utils::menuInstallLocal()
```

After installing the package, it can be loaded with the following code:

```
1 library("KINSIMU")
```

## 3 Functions in the package

In this section, we will introduce the functions contained in the *KINSIMU* package. Each function introduction will consist of four parts: (i) input arguments; (ii) output values; (iii) detailed description of the function's process, presented as algorithms. Code and methods for key steps will be labeled with †, ‡, # and § in the algorithms, followed by detailed explanations. (iv) Examples of the function.

There would be several common abbreviations in this section:

**nl**: number of loci in the panel, used in 'EvaluatePanel()', 'testsimulation()' and 'logLR()' functions;

**ss**: sample size to be simulated in 'pairsimu()', 'pedisimu()' and 'testsimulation()' functions;

**n<sub>a</sub>**: number of alleles on a locus;

**np**: number of individual to be simulated in 'pedisimu()' function;

**stop** in algorithms: Terminate the function without executing any subsequent code.

### 3.1 'EvaluatePanel()'

The function would transfer the input allele frequency data (in a .csv file or a data frame) into form usable for other functions and then calculate several population parameters based on the frequency data for each marker.

**Input arguments** 4 input arguments are needed for the function, one of which is optional.

- **type**: The type of input data, there are 2 acceptable type of input for the argument:
  - 'csv': a .csv file containing the allele frequency data in the panel. In such a file, cell 'A1' should be 'allele' and the other cells in the first row should contain the names of each marker. The names of all detected allele types in the panel should be listed in the first column and the corresponding frequency data on each marker should be contained in the cells according to marker names and allele names. If the input argument 'raremode' is set as 'ISFG' or '1/2N' and the input argument 'Nind' is set as 'lastrow', the sample size in the population survey should be listed in the last row per marker;
  - 'df': a data frame with the similar structure with the aforementioned .csv file, except that the word 'allele' and marker names should be the column names;
- **strpath**: The pathway of the .csv file or the name of the data frame, according to the set of argument **type**.

- **raremode**: The mode of calculation method of rare allele frequency at each locus. There are 3 acceptable type of input for the argument
    - ‘ISFG’: The default input, rare allele frequency data is calculated based on the population survey results, i.e.,  $p_{rare} = (n_a + 1)/(2N + 1)$  if  $n_a$  and  $N$  denote the number of allele types detected and the sample size, respectively;
    - ‘MAF’: The minimum allele frequency is taken as the rare allele frequency;
    - ‘1/2N’: The minimum allele frequency a population survey with a sample size  $N$  can provide is taken as the rare allele frequency;
    - *<numeric>*: A number set as the rare allele frequency, the function would report error if the number  $> 1$ ;
  - **Nind**: (Optional) Mode of sample size data presentation, only need when **raremode** is set as ‘ISFG’ or ‘1/2N’. There are 2 acceptable types of input for the argument:
    - ‘lastrow’: The sample size data is given in the last row of the .csv file;
    - *<integer>*: An integer number can be input and taken as the sample size on all markers.
  - **Th**: The threshold for the difference in allele frequency sum at a locus with 1, to detect data error from rounding error when the frequency sum does not equal 1. Loci exceeding this threshold will be excluded from the calculation.
- 

**Output values** A list of 4 vectors would be output:

- **afmatrix**: A list of  $nl$  data frames that hold information of the allele frequency for each marker. Each data frame is structured with a single column and a row count of  $n_a$ . The row names of the data frames list the names of the alleles.
  - **rare**: A data frame containing rare allele frequencies on each marker;
  - **indicators**: A data frame containing forensic parameters for each marker, which would be detailed introduced;
  - **panelpara**: A data frame containing population parameters for the whole panel, with the form of  $\log_{10}(1 - \text{paramter})$  to avoid displaying the parameters as 1 because they are too close to 1.
- 

**Detailed description** Upon performing an analysis of a specified .csv file or data frame, this function extracts allele frequency data at different loci and evaluate the population parameters for the specific kit using the provided data. The complete procedure is detailed in Algorithm 1.

---

**Algorithm 1:** EvaluatePanel()

---

- 1 Input basic data (*allelefreq*) from a .csv file or a data frame pointed by argument ‘*strpath*’, according to ‘*type*’ set;
  - 2 Conduct formal checks on basic data, **stop** if any error exists;
  - 3 **if** *Nind* = ‘lastrow’ **then**
  - 4 | Take the last row of *allelefreq* data frame as *n\_of\_indi* data frame;
  - 5 | Take the rest rows as *allelefreq* data frame;
  - 6 **else if** *Nind* = *<integer>* **then**
  - 7 | Repeat the *Nind* value as *n\_of\_indi* data frame;
  - 8 **end**
  - 9 Delete loci for which the difference between the sum of allele frequencies and 1 exceeds **Th**;
  - 10 Calculate population parameters for each locus and generate the **indicators** data frame; /\* † \*/
  - 11 Calculate combined population parameters for the whole panel and generate the **panelpara** data frame;
  - 12 Extract/calculate rare frequency and generate the **rare** data frame according to **raremode**; /\* ‡ \*/
  - 13 Divide the *allelefreq* data frame into  $nl$  single data frames and delete rows with 0 values for each;
  - 14 Integrate the  $nl$  data frames into a list named **afmatrix**;
  - 15 Integrate the four output vectors into a list **Result** and output the list.
- 

† The **indicators** data frame contains 12 population parameters computed for each locus:

Table S1: Parameters presented in **indicator** data frame

No.	Name	Note	Equation
1	$n_a$	The number of allele types on a marker	–
2	$H_u$	The unadjusted heterozygosity	$H_u = 1 - S_2^*$
3	$H_a$	The adjusted heterozygosity	$H_a = H_u \times \frac{n_a}{n_a - 1}$
4	MAF	The minimum allele frequency	–
5	PM	The probability of randomly match	$PM = 2S_2^2 - S_4$
6	DP	The discrimination power	$DP = 1 - PM$
7	PED	Probability of exclusion (PE) in duo paternity cases	$PED = 1 - 4S_2 + 4S_3 + 2S_2^2$
8	PET	PE in trio paternity cases	$PET = 1 - 2S_2 + S_3 + 2S_4 - 3S_5 - 2S_2^2 + 3S_2S_3$
9	PEDD	PE in double doubt parentage cases	$PEDD = 1 + 4S_4 - 4S_5 - 3S_6 - 8S_2^2 + 2S_3^2 + 8S_2S_3$
10	RGE	Mean Power of Random Grandparents Excluded**	$RGE = 1 - 4S_2 + 6S_3 - 17S_5 + 28S_6 - 15S_7 - 4S_2^2 + 18S_2S_3$ $- 16S_2S_4 + 5S_2S_5 - 12S_3^2 + 10S_3S_4$
11	RGENM	RGE without the assistant with the mother	$RGENM = 1 - 8S_2 + 16S_3 - 26S_4 + 30S_5 - 15S_6 + 12S_2^2$ $- 24S_2S_3 + 8S_2S_4 + 6S_3^2$
12	PIC	polymorphism information content	$PIC = 1 - S_2 - S_2^2 + S_4$

\*  $S_x$  represents the sum of all frequencies on a locus to the power of  $x$ , i.e.,  $S_x = \sum_i p_i^x$ ;

\*\* RGE is calculated for grand-parentage identification involving four participants: a child, the child’s biological mother and a couple of individuals who are alleged as the child’s grandparents, i.e., the parents of the child’s father.

‡ The data frame **panelpara** includes 6 types of total (T) or cumulative (C) population parameters are calculated for the whole panel: TDP, CPED, CPET, CPEDD, CRGE, CRGENM. The calculation method of these parameters are similar, if set specific parameter on the  $i$ th marker as  $\mathcal{P}_i$ , the total parameter  $\mathcal{TP}$ , then

$$\mathcal{TP} = 1 - \prod_i \mathcal{P}_i \quad (S1)$$

**Examples** Two examples are given based on a .csv file generated by ‘*outputCSV()*’ function. The two examples would result in a same output, which is included in the package as data “*FortytwoSTR*”.

```

1 #A .csv file can be output with FortytwoSTR data
2 path<-tempdir()
3 outputCSV(FortytwoSTR,file.path(path,'data.csv'))
4 #Example 1, 'df' type, by read the csv file into a data frame
5 allele_data <- read.csv(file = file.path(path,'data.csv'), header = TRUE)
6 STR42<- EvaluatePanel(type = 'df', strpath = allele_data,raremode = "ISFG",Nind = "lastrow")
7 #Example 2, 'csv' type, the same evaluation can be done by directly input the csv file
8 STR42_2 <- EvaluatePanel(type = 'csv', strpath = file.path(path,'data.csv'),
9 raremode = "ISFG",Nind = "lastrow")
10 #The data "FortytwoSTR" is generated with these codes.
```

### 3.2 ‘*pairsimu()*’

The function would generate genotype combinations of multiple individual pairs with specific relationships on an autosomal marker, ignoring mutation.

**Input arguments** 4 input arguments are needed for the function:

- **af**: A data frame of 1 column containing the allele frequency data on the marker with the allele names being the row names and ‘Freq’ as column name, which can be generated with function “‘*EvaluatePanel()*’” (e.g., ‘af=FortytwoSTR\$afmatrix[[1]]’) or input with function ‘*data.frame(Freq=c(...))*’ directly;
- **ss**: The sample size of the simulation;

- **delta**: The distribution of IBD or Jacquard coefficient of the target relationship, which should be input as a single row of data with function ‘c()’, the coefficients should be input in order of ‘ $\kappa_0 \rightarrow \kappa_2$ ’ or ‘ $\Delta_1 \rightarrow \Delta_9$ ’;
- **allelename**: The output format setting, There are 2 acceptable types of input for the argument:
  - *FALSE*: The default set, to output the row number of the alleles in the **af** matrix;
  - *TRUE*: To output allele names, i.e., the row names in the **af** matrix.

**Output value** A data frame of 4 columns and ss rows would be output, which denote the genotype combinations of each pair simulated, the first 2 columns denote individual A and the other 2 individual B.

**Detailed description** A directly generating strategy is applied in this function:

**Algorithm 2:** pairsimu()

```

1 Construct the result data frame in the aforementioned form;
2 if The length of delta is 3 (non-inbred cases) then
3   Randomly generate the genotypes (in form of allele position) of ss individual As;           /* † */
4   Generate ss random numbers 1~4 according to delta set;                               /* ‡ */
5   Generate genotypes of ss individual Bs;
6 else if The length of delta is 9 (inbred cases) then
7   Randomly generate the first alleles (in form of allele position) of ss individual As;     /* † */
8   Generate ss random numbers 1~9 according to delta set;                               /* # */
9   Generate the second alleles of ss individual As and genotypes of ss individual Bs;
10 else
11   Report the error of delta setting and stop;
12 end
13 if allelename='TRUE' then
14   Translate the allele positions into allele names;                                     /* § */
15 end
16 Output the result data frame.

```

† These alleles can be randomly generated with ‘sample()’ function, take the 1st allele of individual A as example:

```

1 pop<-1:nrow(af)
2 results $A1<-sample(x=pop,size=ss,replace=TRUE,prob=af$Freq)

```

‡ For outbred relationships with  $\kappa = \{ \kappa_0, \kappa_1, \kappa_2 \}$ , there are 4 types of possible IBD genotype of individual B, (i)  $x_I y_I$ , (ii)  $b_I x_I$ , (iii)  $a_I x_I$ , and (iv)  $a_I b_I$ , the corresponding probabilities of which are  $\kappa_0$ ,  $\kappa_1/2$ ,  $\kappa_1/2$  or  $\kappa_2$ , respectively. Among these IBD genotypes,  $b_I x_I$  can be written as  $x_I b_I$ , which would not affect any of the following calculations. Thus, the 1st allele of individual B would be IBD to the 1st of individual A if situation (iii) or (iv) happens, while his/her 2nd allele would be IBD to the 2nd of individual A if situation (ii) or (iv) happens. In other words, the genotypes of individual Bs can be generated according to ‘delta’ value with the following code:

```

1 # Randomly generate individual A's alleles
2 results $A1<-sample(x=pop,size=ss,replace=TRUE,prob=af$Freq)
3 results $A2<-sample(x=pop,size=ss,replace=TRUE,prob=af$Freq)
4 # Generate random numbers (RN), with value of 1:4 according to delta distribution
5 # Each number stands for a situation of individual B's IBD genotype
6 RN<-sample(x=1:4,size=ss,replace=TRUE,prob=c(delta[1],delta[2]/2,delta[2]/2,delta[3]))
7 # T1: numbers larger than 2 in RNs, i.e., situation (iii) or (iv) happens
8 T1<-as.double(RN>2)
9 # T2: even numbers in RNs, i.e., situation (ii) or (iv) happens
10 T2<-as.double(RN%%2==0)
11 # Transfer situations into individual B's alleles
12 results $B1<-sample(x=pop,size=ss,replace=TRUE,prob=af$Freq)*(1-T1)+results$A1*T1
13 results $B2<-sample(x=pop,size=ss,replace=TRUE,prob=af$Freq)*(1-T2)+results$A2*T2

```

# For inbred relationships, the two alleles of individual A may be IBD to each other, i.e., only the first alleles of each individual A can be generated randomly based on the allele frequency data and other 3 alleles should be generated according to the  $\Delta$  distribution, which is more complex compared to outbred cases. The corresponding code can be written as follows:

```

1 results $A1<-sample(x=pop,size=ss,replace=TRUE,prob=af$Freq)
2 RN<-sample(x=1:11,size=ss,replace=TRUE,prob=c(delta[1],delta [2], delta [3], delta [4], delta [5]/2, delta [5]/2, delta [6], delta [7], delta [8]/2, delta [8]/2,
delta [9]))
3 T0<-as.double(RN>4)
4 T1A<-as.double(RN%in%c(1,3,5,8,9))
5 T1B<-as.double(RN==6)
6 T2A<-as.double(RN%in%c(1,8,10))
7 T2B<-as.double(RN%in%c(2,5,6,7))
8 results $A2<-sample(x=pop,size=ss,replace=TRUE,prob=af$Freq)*T0+results$A1*(1-T0)
9 results $B1<-sample(x=pop,size=ss,replace=TRUE,prob=af$Freq)*(1-T1A-T1B)+results$A1*T1A+results$A2*T1B
10 results $B2<-sample(x=pop,size=ss,replace=TRUE,prob=af$Freq)*(1-T2A-T2B)+results$A2*T2A+results$B1*T2B

```

§ If argument **allelename** is set as TRUE, the allele position can be translated to allele names with the following codes:

```

1 an<-as.data.frame(as.numeric(row.names(af)))
2 results $A1<-an[results$A1,]
3 results $A2<-an[results$A2,]
4 results $B1<-an[results$B1,]
5 results $B2<-an[results$B2,]

```

**Example** Three examples are given, simulating 3 types of individual pairs based on the 1st STR in the 42 ones in ‘FortytwoSTR’ data set, setting ss=10,000:

```

1 # Extract allele frequency on the locus
2 af = FortytwoSTR$afmatrix[[1]]
3 # simulating 10,000 unrelated pairs
4 a<-pairsimu(af = af, ss = 10000, delta = c(1,0,0) , allelename = FALSE)
5 # simulating 10,000 parent-child pairs
6 b<-pairsimu(af = af, ss = 10000, delta = c(0,1,0) , allelename = FALSE)
7 # simulating 10,000 full-sibling pairs
8 c<-pairsimu(af = af, ss = 10000, delta = c(0.25,0.5,0.25) , allelename = FALSE)

```

### 3.3 ‘pedisimu()’

The function would genotype combinations of multiple pedigrees with specific relationships on an autosomal marker.

**Input arguments** 7 input arguments are needed for the function, 2 of which are optional.

- **af**: A data frame of similar to **af** in ‘pairsimu()’ function;
- **ss**: The sample size of the simulation;
- **pedi**: a data frame containing the pedigree structure information, with 3 columns (“Person”, “Father” and “Mother”) and *np* rows;
- **random\_name**: The name of random individual, with a default of "RI";
- **muf**: (Optional) father-child mutation rate, with default set of 0, only need if **allelename** set as *TRUE*;
- **mum**: (Optional) mother-child mutation rate, with default set of 0, only need if **allelename** set as *TRUE*;
- **allelename**: Similar to **allelename** in ‘pairsimu()’ function.

**Output value** A data frame of *ss* rows and  $2 \times np$  columns. Each pair of columns contains alleles of an individual, with the individuals sorted in the same order as in the **pedi** data frame.

**Detailed description** Genotype data is generated sequentially according to Algorithm 3:

---

**Algorithm 3:** `pedisimu()`

---

```

1 Conduct formal checks on the data frame pedi, and stop if any error exists;
2 Extract  $np$  as the row number of pedi data frame;
3 Construct result data frame with  $ss$  rows and  $2 \times np$  columns;
4 for  $i$  in  $1:np$  do
5   if There is no individual in “Person” column named identical as the father of  $i$ th individual then
6     | Randomly generate the  $i$ th individual’s paternal allele according to allelename set;           /* † */
7   else
8     | Set  $f$ =the row number of the  $i$ th individual’s father in “Person” column;
9     | if  $f \geq i$  then
10    | | Report error “Father of the  $i$ th individual should be defined before him/her” and stop;
11    | end
12    | Generate the  $i$ th individual’s paternal allele from the  $f$ th individual’s genotype;           /* ‡ */
13    | if  $muf > 0$  then
14    | | Randomly change the alleles according to  $muf$  set;                               /* # */
15    | end
16  end
17  Generate the maternal allele of the  $i$ th individual with similar process considering mum setting;
18 end

```

---

† Alleles without parent setting can be generated with ‘*sample()*’ function, according to the **allelename** set. Take the paternal allele of the  $i$ th individual as example:

```

1 if (isTRUE(allelename)) { results [,2*i-1]←sample(x=as.numeric(row.names(af)),size=ss,replace=TRUE,prob=af$Freq)
2 } else {pop←1:nrow(af)
3   results [,2*i-1]←sample(x=pop,size=ss,replace=TRUE,prob=af$Freq)
4 }

```

‡ If father or mother of the  $i$ th individual is set, the corresponding allele can be generated according to the genetic law, i.e., the two alleles of the father or mother have equivalent probabilities to be inherited. Take the paternal inheritance of the offspring as example:

```

1 RN←sample(x=c(0,1),size=ss,replace=TRUE,prob=c(0.5,0.5))
2 results [,2*i-1]←results [,2*f-1]*RN+results [,2*f]*(1-RN)

```

# To perform mutation, the original alleles of the offspring would be added by a random number within set  $\{-1,0,1\}$  according to the mutation rate:

```

1 if ( $muf > 0$ ) { results [,2*i-1]=results [,2*i-1]+sample(x=c(-1,0,1), size=ss, replace=TRUE,prob=c( $muf/2,1-muf,muf/2$ ))}

```

**Example** An example is given to simulate 10,000 first cousin pedigree (as listed in data ‘**pediexample**’ included in the package), based on the first locus in data ‘*FortytwoSTR*’:

```

1 pedi←pediexample
2 af←FortytwoSTR$afmatrix[[1]]
3 pedisimu(af=af, ss=10000,pedi=pedi)

```

### 3.4 ‘*LRparas()*’

The function would count or calculate parameters used in the calculation of different pairwise LR.

**Input arguments** 7 input arguments are needed for the function

---

- **AB**: A data frame of 4 columns containing the genetic information of the participants, each row denote a single pair, and the 4 columns containing alleles  $a$ ,  $b$ ,  $c$  and  $d$ , respectively. The data frame can be generated with functions ‘*pairsimu()*’ or ‘*pedisimu()*’, or can be directly input with function ‘*data.frame()*’;
  - **af**: Allele frequency data, There are 2 acceptable types of input for the argument:
    - *NULL*: The default set;
    - *<data frame>*: input a data frame similar to **af** in ‘*pairsimu()*’ function;
  - **rare**: Rare frequency data, there are 2 acceptable types of input for the argument:
    - *NULL*: The default set;
    - *<numeric>*: A number of rare frequency;
  - **stepwisePI**: Whether stepwise mutation model should be considered when calculating paternity index, there are 2 acceptable types of input for the argument:
    - *FALSE*: The default set, take mutation rate as PI if no allele is shared between the two individuals;
    - *TRUE*: Calculate PI considering stepwise mutation model in paternity tests;
  - **bred**: Whether  $J_1 \sim J_6$  should be calculated, there are 2 acceptable types of input for the argument:
    - *FALSE*: The default set, output 2 ratios: (i)  $\Pr(E|J_8)/\Pr(E|J_9)$ , i.e., paternity index which is labeled as ‘*PI<sub>nomu</sub>*’ in the output data frame; (ii)  $\Pr(E|J_7)/\Pr(E|J_9)$ , i.e., likelihood ratio in personal identification which is labeled as ‘*LR<sub>id</sub>*’ in the output data frame;
    - *TRUE*: If so, the other 6 ratios in Eq. (15) would be output, labeled as ‘*FDI*’  $\rightarrow$  ‘*FD6*’, i.e., factors of  $\Delta_1 \rightarrow \Delta_6$ ;
  - **mu**: The mutation rate if paternity index is calculated, with a default of 0.002;
  - **allelename**: The data type of **AB** data frame, there are 2 acceptable types of input for the argument:
    - *FALSE* To treat the values in **AB** data frame as row numbers in **af** data frame;
    - *TRUE* To treat the values in **AB** data frame as allele names (row names) in **af** data frame.
- 

**Output value** A data frame of **ss** rows and multiple columns, containing the combined identity by state (CIBS) score, the 8 or 2 ratios needed in Eq. (15) or Eq. (16) in the main text, respectively, as well as PI considering mutation.

**Detailed description** The function is performed as Algorithm 4. It can be seen that all these parameters are calculated simultaneously, as a result, LR of the **ss** pairs can be calculated simultaneously on a single marker.



---

**Algorithm 4: LRparas()**

---

```
1 Conduct formal checks on the data frame AB, and stop if any error exists;
2 Calculate  $\mathbb{1}_{ac}$ ,  $\mathbb{1}_{ad}$ ,  $\mathbb{1}_{bc}$  and  $\mathbb{1}_{bd}$  based on AB data frame; /* † */
3 Calculate IBS score and set as the first column of the output result data frame; /* ‡ */
4 if (af='NULL') OR (allelename='TRUE' AND rare=INULL) then
5   if stepwisePI or bred is set as TRUE then
6     Report error "Please input the frequency data" and stop;
7   end
8   Output the result data frame; /* No frequency data for LR calculation */
9 else
10  Extract  $p_c$  and  $p_d$ ; /* # */
11  Calculate LRid and PInomu (PI not considering mutation) as the 2nd and 3rd columns of the result data frame,
    according to Eq. (16) in main text;
12  if All values in PInomu column >0 then
13    Set PImu=PInomu as the 4th column of the result data frame;
14  else if stepwisePI is set as TRUE /* § */ then
15    Calculate the 4 absolute differences between the two individual's alleles ( $a \rightarrow c$ ,  $a \rightarrow d$ ,  $b \rightarrow c$ , and  $b \rightarrow d$ ),
    transfer non-integer-step mutations into numbers larger than 10000;
16    Choose the minimum in the 4 differences as the minimum mutation step ( $s$ );
17    Calculate  $d_{Ac}$  and  $d_{Ad}$  as the dosage of alleles that can mutate into  $c$  and  $d$  with  $s$  steps;
18    if  $s=0$  then
19      PImu=PInomu;
20    else if  $s > 10000$  then
21      PImu= $\mu$ , which is defined by input argument mu;
22    else
23       $PImu = \mu \times 10^{1-s} \times \left( \frac{d_{Ac}}{8p_c} + \frac{d_{Ad}}{8p_d} \right)$ ;
24    end
25  else
26    Calculate PImu, by taking mu as PI when no allele is shared between the two individuals;
27  end
28  if bred is set as TRUE then
29    Extract  $p_a$ , calculate  $\mathbb{1}_{ab}$  and  $\mathbb{1}_{cd}$ ;
30    Calculate the other 6 ratios in Eq. (15) as the 5th~10th columns of the result data frame;
31  end
32  Output the result data frame
33 end
```

---

† The  $\mathbb{1}$  parameters can be calculated using '*as.double()*' function, take  $\mathbb{1}_{ac}$  as example:

```
1 ac<-as.double(AB[,1]==AB[,3])
```

‡ The IBS score is a parameter evaluating the similarity of two individual's genotypes. It equals to 2 if the genotypes are identical (i.e.,  $\mathbb{1}_{ac}\mathbb{1}_{bd} + \mathbb{1}_{bc}\mathbb{1}_{ad} > 0$ ), to 0 if there is no allele sharing between them (i.e.,  $\mathbb{1}_{ac} + \mathbb{1}_{bd} + \mathbb{1}_{bc} + \mathbb{1}_{ad} = 0$ ), and to 1 otherwise. Thus, the score can be calculated with following codes:

```
1 ibs=as.double(ac+ad+bc+bd>0)+as.double(ac*bd+ad*bc>0)
```

# The frequency value can be extracted from **af** data frame according to **allelename** setting, then, frequency of rare alleles should be replaced by **rare** value. Take  $p_c$  as example:

```
1 if (isTRUE(allelename)) {pc<-af[as.character(AB[,3]),]} else {pc<-af[AB[,3],]}
2 pc[is.na(pc)]<-rare
```

§ When using STR markers in parentage testing, the relatively high mutation rate of these markers leads to a greater likelihood that a true parent-child pair may not share any allele. In such cases, it is important to

calculate the PI considering mutation in order to prevent the erroneous exclusion of parentage. Replication slippage is the primary cause of mutation for STR markers, and a model known as the “stepwise mutation model” has been developed, considering that the probability of mutating  $s+1$  steps as being one-tenth of the probability of mutating  $s$  steps. Thus, the probability of allele  $x$  mutated to allele  $x+s$  equal to  $0.5 \times \mu \times \left(\frac{1}{10}\right)^{s-1}$ , where the first “0.5” denote the probability of mutating longer or shorter. And the calculation of PI considering mutation can be performed with the following codes:

```

1 # Calculate the absolute difference between the two individual's alleles , take abs(a-c) as example
2 if ( is.TRUE(allelename) ) {
3   acm=abs(AB[,1]-AB[,3])+abs(AB[,1]-AB[,3])%%1*100000
4 } else {
5   an<-as.data.frame(as.numeric(row.names(af)))
6   acm=abs(an[AB[,1],]-an[AB[,3],])+abs(an[AB[,1],]-an[AB[,3],]) %%1*100000
7 }
8 # Choose the minimum mutation steps
9 steps<-pmin(acm,adm,bcm,bdm)
10 # dc and dd calculation
11 dc<-as.double(acm==steps)+as.double(bcm==steps)
12 dd<-as.double(adm==steps)+as.double(bdm==steps)
13 #PImu calculation
14 PImu<-as.double(steps==0)*para$PInomu+ # ignore mutation if step=0, i.e., there is at least one sharing allele between the two individuals
15 as.double(steps>10000)*mu+ # take mutation rate as PI if there is no integer step of mutation
16 as.double(steps>0 & steps<10000)*mu*10^(1-steps)*(dc/pc+dd/pd)/8 # step wise model of mutation

```

**Example** An example is given, simulating and calculating parameters for 10,000 parent-child pairs

```

1 af = FortytwoSTR$afmatrix[[1]]
2 AB = pairsimu(af = af, ss = 10000,delta = c(0,1,0) ,allelename = FALSE)
3 LRelements<-LRparas(AB=AB, af=af, rare=FortytwoSTR$rare[1],allelename=FALSE,stepwisePI=TRUE,bred=TRUE)

```

The following 4 functions are utilized to compute more intricate LR for specific identification across multiple cases on a single marker:

### 3.5 ‘trioPI()’

The function would Calculate LR in standard trio cases, where 3 participants being available, a child (C), one of his/her confirmed parent (TP), as well as an individual who is unrelated to TP and alleged to be specific relative of the child (AR). Usually, AR is the alleged father of C, as in standard trio paternity testing. Null hypothesis, i.e., that the alleged participant is unrelated to the child, is taken as Hd. Inbreeding factors are not considered.

**Input arguments** 9 input arguments are needed for the function

- **AR**: A data frame of 2 columns containing the genetic information of ARs;
- **C**: A data frame of 2 columns containing the genetic information of Cs;
- **TP**: A data frame of 2 columns containing the genetic information of ARs;
- **af**: A data frame similar to **af** in ‘pairsimu()’ function;
- **rare**: The frequency of rare alleles, which is similar to **rare** in ‘LRparas()’ function;
- **allelename**: The data type of **AR**, **C** and **TP** data frames, similar to **allelename** in ‘LRparas()’ function;
- **muAtoC**: Mutation rate from AR to C, if AR is alleged to be a parent of C. With a default set of 0.002, please note that mistakes would be introduced if the mutation rate is larger than 0.2;
- **muTtoC**: Mutation rate from TP to C, with a default of 0.002/3.5, please note that mistakes would be introduced if the mutation rate is larger than 0.2;
- **kappa1**: The  $\kappa_1$  between AR and C under Hp, with a default set of 1, i.e., trio paternity testing.

**Output value** A data frame of **ss** rows and 1 columns would be output, containing the  $\log_{10}$ LR of each case.

**Detailed description** The function is performed as Algorithm 5.

---

**Algorithm 5:** trioPI()

---

```

1 Conduct formal checks on the 3 genotype data frames, and stop if any error exists;
2 Extract  $p_c$  and  $p_d$  with similar method to code in “#” of ‘LRparas()’ function;
3 Calculate the four  $d$  parameters in Eq. (25) of the main text, each as the summation of two  $\mathbb{1}$  parameters calculated
  similar to code in “†” of ‘LRparas()’ function;
4 if  $kappa1 < 1$  then
5   if  $d_{TPc} + d_{TPd} > 0$  for all cases then
6     Calculate results directly according to Eq. (25) in the main text:
7      $LR = \frac{\kappa_1 d_{TPc} d_{ARd} + \kappa_1 d_{TPd} d_{ARc}}{2d_{TPc} p_d + 2d_{TPd} p_c} + (1 - \kappa_1)$ 
8   else
9     Calculate results considering mutation from TP to C;
10  end
11 else
12  if  $d_{TPc} d_{ARd} + d_{TPd} d_{ARc} > 0$  for all cases then
13    Calculate results directly according to Eq. (27) in the main text:
14     $PI_{trio} = \frac{d_{Mc} d_{AFd} + d_{Md} d_{AFc}}{2d_{Mc} p_d + 2d_{Md} p_c}$  /* kappa=1, i.e., trio paternity testing */
15  else
16    Calculate results considering mutations from both TP and AR to C.
17  end
18 end

```

---

**Example** Examples of two type of identifications are given: trio paternity test and avuncular test with the assistance of the child’s mother:

```

1 # Three types of pedigrees are simulated: pedi1: father–mother–child; pedi2: random male–mother–child; and pedi3: uncle–mother–child
2 pedi1 <- data.frame(Person=c("F","M","C"),Father=c("RI","RI","F"),Mother=c("RI","RI","M"))
3 pedi2 <- data.frame(Person=c("F","M","C"),Father=c("RI","RI","RI"),Mother=c("RI","RI","M"))
4 pedi3 <- data.frame(Person=c("GF","GM","AR","F","M","C"),
5   Father=c("RI","RI","GF","GF","RI","F"),
6   Mother=c("RI","RI","GM","GM","RI","M"))
7 # Two types of LRs are calculated: PI_1 and PI_2: Paternity index for pedi1 and pedi 2; AI_1 and AI_2: Avuncular index in trio cases for pedi2
  and pedi3.
8
9 PI_1=PI_2=AI_1=AI_2=data.frame(Log10CLR=rep(0,10000))
10 # Simulation are carried out based on the frequency data of the 42 STRs in FortytwoSTR dataset setting sample size as 10,000
11 for (i in 1:42) {
12   Genotype1<-pedisimu(af = FortytwoSTR$afmatrix[[i ]], ss = 10000,pedi = pedi1)
13   PI_1<-PI_1+trioPI(AR=Genotype1[,1:2],TP=Genotype1[,3:4],
14   C=Genotype1[,5:6],af=FortytwoSTR$afmatrix[[i ]],
15   rare=FortytwoSTR$rare[i])
16   Genotype2<-pedisimu(af = FortytwoSTR$afmatrix[[i ]], ss = 10000,pedi = pedi2)
17   PI_2<-PI_2+trioPI(AR=Genotype2[,1:2],TP=Genotype2[,3:4],
18   C=Genotype2[,5:6],af=FortytwoSTR$afmatrix[[i ]],
19   rare=FortytwoSTR$rare[i])
20   AI_2<-AI_2+trioPI(AR=Genotype2[,1:2],TP=Genotype2[,3:4],
21   C=Genotype2[,5:6],af=FortytwoSTR$afmatrix[[i ]],
22   rare=FortytwoSTR$rare[i],kappa1=0.5)
23   Genotype3<-pedisimu(af = FortytwoSTR$afmatrix[[i ]], ss = 10000,pedi = pedi3)
24   AI_1<-AI_1+trioPI(AR=Genotype3[,5:6],TP=Genotype3[,9:10],
25   C=Genotype3[,11:12],af=FortytwoSTR$afmatrix[[i ]],
26   rare=FortytwoSTR$rare[i],kappa1=0.5)
27 }
28 #histogram of the final results
29 xmin1<-floor(min(min(PI_1$Log10CLR),min(PI_2$Log10CLR)))
30 xmax1<-ceiling(max(max(PI_1$Log10CLR),max(PI_2$Log10CLR)))
31 xmin2<-floor(min(min(AI_1$Log10CLR),min(AI_2$Log10CLR)))
32 xmax2<-ceiling(max(max(AI_1$Log10CLR),max(AI_2$Log10CLR)))
33 par(mfrow = c(2, 2))
34 hist(PI_1$Log10CLR,xlab = expression(log[10]-CPI),main = "True parentage cases",
35   xlim = c(xmin1,xmax1), col = "blue")

```

```

36 hist (AI_1$Log10CLR,xlab = expression(log[10]~CAI),main = "True avuncular cases",
37 xlim = c(xmin2,xmax2), col = "blue")
38 hist (PI_2$Log10CLR,xlab = expression(log[10]~CPI),main = "False pedigree in parentage cases",
39 xlim = c(xmin1,xmax1), col = "red")
40 hist (AI_2$Log10CLR,xlab = expression(log[10]~CAI),main = "False pedigree in avuncular cases",
41 xlim = c(xmin2,xmax2), col = "red")

```

### 3.6 ‘IICAL()’

The function would calculate incest index ( $II_\varphi$ ), i.e., the ratio for a parent-child pair between the probability that the child’s other parent is a relative of the present parent to the probability that the child’s parents are unrelated.

**Input arguments** 6 input arguments are needed for the function

- **Parent:** A data frame of 2 columns containing the genetic information of the present parent;
- **Child:** A data frame of 2 columns containing the genetic information of Cs;
- **af:** A data frame similar to **af** in ‘*pairsimu()*’ function;
- **rare:** The frequency of rare alleles, which is similar to **rare** in ‘*LRparas()*’ function;
- **allelename:** The data type of **Parent** and **Child** data frames, similar to **allelename** in ‘*LRparas()*’ function;
- **phi:** The kinship coefficient  $\varphi$  between the two parents under Hp, with a default set of 0.25, i.e., assuming that they are parent-child pair or full-siblings.

**Output value** A data frame of **ss** rows and 2 columns would be output, containing two parameters used in such cases: (i) Ngs: the state of genotype similarity (gs) between the parent-child pair, i.e., whether both of the child’s two alleles can be inherited from the present parent; (ii) the  $\log_{10}II_\varphi$  of each case.

**Detailed description** The function is performed as Algorithm 6.

#### Algorithm 6: IICAL()

- 1 Conduct formal checks on the 2 genotype data frames, and **stop** if any error exists;
- 2 Extract or calculate the two **d** parameters and two **p** parameters in Eq. (20) of the main text with codes similar to corresponding ones in ‘*LRparas()*’ function:  $II_\varphi = \frac{2\varphi d_{Ac}d_{Ad}}{d_{Ac}p_d + d_{Ad}p_c} + (1 - 2\varphi)$ ;
- 3 Calculate Ngs with code “Ngs<-1-as.double(dc\*dd==0)”;
- 4 Calculate  $II_\varphi$  with code “IIphi<-log10(2\*phi\*dc\*dd/(dc\*pd+dd\*pc)+1-2\*phi)”;
- 5 Translate the error values (for with the parent cannot provide any of the child’s alleles) into 1-2 $\varphi$ .

Note that if there is no allele sharing between the two participants,  $II_\varphi$  cannot be calculated directly according to Eq. (20) in the main text due to the 0 value of the denominator. In this case,  $(1 - 2\varphi)$  would be the output.

**Example** Two examples are given, simulating and calculating  $CII_{0.25}$  based on the 42 STRs in *FortytwoSTR* dataset for 10,000 mother-child pairs when Hp and Hd in the formula is true, respectively.

```

1 # Construct the pedi data.frame for incest cases
2 pedi<-data.frame(Person=c("F","M","C"),Father=c("RI","F","F"),Mother=c("RI","RI","M"))
3 II_1=II_2=data.frame(Ngs=rep(0,10000),IIphi=rep(0,10000))
4 for (i in 1:42) {
5 # Simulate 10,000 mother-child pairs with father-daughter incest with pedisimu() function
6 Genotype1<-pedisimu(af = FortytwoSTR$afmatrix[[i ]], ss = 10000,pedi = pedi)
7 II_1<-II_1+IICAL(Parent = Genotype1[,3:4],Child = Genotype1 [,5:6], af=FortytwoSTR$afmatrix[[i ]],
8 rare=FortytwoSTR$rare[i ][1,1], phi=0.25)
9 #Simulate 10,000 non-inbred mother-child pairs with pairsimu() function
10 Genotype2<-pairsimu(af = FortytwoSTR$afmatrix[[i ]], ss = 10000,delta = c(0,1,0) , allelename = FALSE)
11 II_2<-II_2+IICAL(Parent = Genotype2[,1:2],Child = Genotype2 [,3:4], af=FortytwoSTR$afmatrix[[i ]],
12 rare=FortytwoSTR$rare[i ][1,1], phi=0.25)
13 }
14 # histograms of CII distributions in the two groups
15 xmin<-floor(min(min(II_1$IIphi),min(II_2$IIphi)))
16 xmax<-ceiling(max(max(II_1$IIphi),max(II_2$IIphi)))

```

```

17 par(mfrow = c(1, 2))
18 hist (II_2$IIphi, xlab = expression(log[10]-CII), main = "Non-inbred cases",
19 xlim = c(xmin,xmax), col = "red")
20 hist (II_1$IIphi, xlab = expression(log[10]-CII), main = "Inbred cases",
21 xlim = c(xmin,xmax), col = "blue")

```

### 3.7 ‘LRhsip()’

The function would calculate LR for cases in which a pair of siblings (labeled as A and B, with genotype  $ab$  and  $cd$ , respectively) and one of their identical parent participated (labeled as P with genotype  $ef$ ).  $H_p$  and  $H_d$  are set as "the other parents of the two siblings are specific related" and "the other parents of them are unrelated", respectively. Inbreeding factors are not taken into consideration.

**Derivation of LR in the identification** Regard sibling B as individual B, and the other two individuals as R in Eq. (21) of the main text. Similar to “standard” non-inbred trio cases in section 2.4.2 of the main text, the relationship between P and A would remain unchanged in both hypotheses if not considering inbreeding factors, thus, similarly

$$LR = \frac{\sum_j [\Pr(B \equiv \mathcal{G}_j | A, P, H_p) \times \Pr(\mathcal{G}_j \equiv cd)]}{\sum_k [\Pr(B \equiv \mathcal{G}_k | A, P, H_d) \times \Pr(\mathcal{G}_k \equiv cd)]} \quad (S2)$$

Under  $H_d$ , the non-participated parent of sibling B is unrelated to both P and sibling A, the allele he/she passed to sibling B must be  $x_I$  from the perspective of IBD alleles. Thus, there are two types of sibling B’s IBD genotype with equal probabilities,  $e_I x_I$  and  $f_I x_I$ .

Under  $H_p$ , the distribution of sibling B’s IBD genotype can be derived as follows: (i) derive the IBD genotype distribution of sibling A’s non-participated parent (labeled as NPA); (ii) derive the IBD genotype distribution of sibling B’s non-participated parent (labeled as NPB); and (iii) derive sibling B’s IBD genotype.

$$\begin{aligned} \Pr(B \equiv \mathcal{G}_j | A, P, H_p) &= \sum_x [\Pr(NPA \equiv \mathcal{G}_x | A, P, H_p) \times \Pr(B \equiv \mathcal{G}_j | NPA \equiv \mathcal{G}_x, A, P, H_p)] \\ \Pr(B \equiv \mathcal{G}_j | NPA \equiv \mathcal{G}_x, A, P, H_p) &= \sum_y [\Pr(NPB \equiv \mathcal{G}_y | NPA \equiv \mathcal{G}_x, A, P, H_p) \times \Pr(B \equiv \mathcal{G}_j | NPB \equiv \mathcal{G}_y, NPA \equiv \mathcal{G}_x, A, P, H_p)] \end{aligned} \quad (S3)$$

(i) If not considering mutation, the IBD genotype of NPA must be  $a_I x_I$  or  $b_I x_I$ , the probabilities of which can be calculated with Bayes’ rules:

$$\Pr(NPA \equiv a_I x_I | A, P, H_p) = \frac{\Pr(NPA \equiv a_I x_I | P, H_p) \times \Pr(A | NPA \equiv a_I x_I, P, H_p)}{\Pr(NPA \equiv a_I x_I | P, H_p) \times \Pr(A | NPA \equiv a_I x_I, P, H_p) + \Pr(NPA \equiv b_I x_I | P, H_p) \times \Pr(A | NPA \equiv b_I x_I, P, H_p)} \quad (S4)$$

Where  $\Pr(NPA \equiv a_I x_I | P, H_p) = \Pr(NPA \equiv a_I x_I) = p_a$  under non-inbred assumption. And because of the same assumption, the allele  $b$  of sibling A must inherited from P, i.e.,  $\Pr(A | NPA \equiv a_I x_I, P, H_p) \propto d_{pb}$ . Thus,

$$\begin{cases} \Pr(NPA \equiv a_I x_I | A, P, H_p) = \frac{p_a d_{pb}}{p_a d_{pb} + p_b d_{pa}} \\ \Pr(NPA \equiv b_I x_I | A, P, H_p) = \frac{p_b d_{pa}}{p_a d_{pb} + p_b d_{pa}} \end{cases} \quad (S5)$$

(ii) Under  $H_p$ , NPB should be unrelated to P and the relationship between him/her with sibling A should happened through NPA, i.e., the genotype of P and sibling A should be useless for the IBD genotype inference of NPB when the IBD genotype of NPA is given, i.e.,  $\Pr(NPB \equiv \mathcal{G}_y | NPA \equiv \mathcal{G}_x, A, P, H_p) = \Pr(NPB \equiv \mathcal{G}_y | NPA \equiv \mathcal{G}_x, H_p)$ .

If no inbreeding factor is considered, the relationship between NPA and NPB can be described by  $\kappa = \{\kappa_0, \kappa_1, \kappa_2\}$ . It can be seen that, if  $NPB \equiv a_I x_I$  as example, there can be 2 types of NPB's IBD genotype,  $a_I x_I$  and  $x_I y_I$ , with probabilities of  $\kappa_2 + \kappa_1/2 = 2\varphi$  and  $\kappa_0/2 + \kappa_1/2 = (1 - 2\varphi)$ , respectively, if  $\varphi$  denotes the kinship coefficient between NPA and NPB; Similarly, if  $NPB \equiv b_I x_I$  as example, NPB's IBD genotype can be  $b_I x_I$  or  $x_I y_I$ , with probabilities of  $2\varphi$  and  $(1 - 2\varphi)$ , respectively;

(iii) Similar to the inference of NPB's IBD genotype, NPA and sibling A's genotypes are useless for the IBD genotype inference of sibling B when the IBD genotype of NPB and P are given, i.e.,

$$\Pr(B \equiv \mathcal{G}_j | NPB \equiv \mathcal{G}_y, NPA \equiv \mathcal{G}_x, A, P, H_p) = \Pr(B \equiv \mathcal{G}_j | NPB \equiv \mathcal{G}_y, A, H_p) \quad (S6)$$

In summary, there can be 6 types of sibling B's IBD genotype, the inference process is listed in Table S2:

Table S2: Sibling B's IBD genotype

B	NPA	$\Pr(NPA \equiv a_I x_I   A, P, H_p)$	NPB	$\Pr(NPB \equiv \mathcal{G}_y   NPA \equiv \mathcal{G}_x, H_p)$	$\Pr_1^*$	Total Probability
$a_I e_I$	$a_I x_I$	$\frac{p_a d_{pb}}{p_a d_{pb} + p_b d_{pa}}$	$a_I x_I$	$2\varphi$	$\frac{1}{4}$	$\frac{\varphi p_a d_{pb}}{2p_a d_{pb} + 2p_b d_{pa}}$
$a_I f_I$	$a_I x_I$	$\frac{p_a d_{pb}}{p_a d_{pb} + p_b d_{pa}}$	$a_I x_I$	$2\varphi$	$\frac{1}{4}$	$\frac{\varphi p_a d_{pb}}{2p_a d_{pb} + 2p_b d_{pa}}$
$b_I e_I$	$b_I x_I$	$\frac{p_b d_{pa}}{p_a d_{pb} + p_b d_{pa}}$	$b_I x_I$	$2\varphi$	$\frac{1}{4}$	$\frac{\varphi p_b d_{pa}}{2p_a d_{pb} + 2p_b d_{pa}}$
$b_I f_I$	$b_I x_I$	$\frac{p_b d_{pa}}{p_a d_{pb} + p_b d_{pa}}$	$b_I x_I$	$2\varphi$	$\frac{1}{4}$	$\frac{\varphi p_b d_{pa}}{2p_a d_{pb} + 2p_b d_{pa}}$
$e_I x_I$	$a_I x_I$	$\frac{p_a d_{pb}}{p_a d_{pb} + p_b d_{pa}}$	$a_I x_I$	$2\varphi$	$\frac{1}{4}$	$\frac{1-\varphi}{2}$
	$b_I x_I$	$\frac{p_b d_{pa}}{p_a d_{pb} + p_b d_{pa}}$	$x_I y_I$	$1 - 2\varphi$	$\frac{1}{2}$	
$f_I x_I$	$a_I x_I$	$\frac{p_a d_{pb}}{p_a d_{pb} + p_b d_{pa}}$	$a_I x_I$	$2\varphi$	$\frac{1}{4}$	$\frac{1-\varphi}{2}$
	$b_I x_I$	$\frac{p_b d_{pa}}{p_a d_{pb} + p_b d_{pa}}$	$x_I y_I$	$1 - 2\varphi$	$\frac{1}{2}$	

\*  $\Pr_1 = \Pr(B \equiv \mathcal{G}_j | NPB \equiv \mathcal{G}_y, A, H_p)$ ;

With calculation similar to Eq. (24) in the main text, it can be derived that

$$LR = \frac{\varphi p_a d_{pb} (\mathbb{1}_{ac} d_{pd} + \mathbb{1}_{ad} d_{pc}) + \varphi p_b d_{pa} (\mathbb{1}_{bc} d_{pd} + \mathbb{1}_{bd} d_{pc})}{(d_{pa} p_b + d_{pb} p_a) (d_{pc} p_d + d_{pd} p_c)} + (1 - \varphi) \quad (S7)$$

According to the equation, the function 'LRhsip()' is constructed.

**Input arguments** 7 input arguments are needed for the function

- **A**: A data frame of 2 columns containing the genetic information of individual As;
- **B**: A data frame of 2 columns containing the genetic information of individual Bs;
- **P**: A data frame of 2 columns containing the genetic information of the identical parents;
- **af**: A data frame similar to **af** in 'pairsimu()' function;
- **rare**: The frequency of rare alleles, which is similar to **rare** in 'LRparas()' function;

- **allelename**: The data type of **A**, **B** and **P** data frames, similar to **allelename** in ‘*LRparas()*’ function;
- **phi**: The kinship coefficient  $\varphi$  between the non-participant parents of the two half-siblings under Hp, with a default set of 0.5, i.e., assuming that individual A is a full-sibling of individual B.

**Output value** A data frame of **ss** rows and 1 columns would be output, containing the  $\log_{10}$ LR of each case.

**Detailed description** The function is performed as Algorithm 7.

---

**Algorithm 7: LRhsip()**

---

- 1 Conduct formal checks on the 3 genotype data frames, and **stop** if any error exists;
  - 2 Extract or calculate the **d** parameters,  $\mathbb{1}$  parameters, and **p** parameters in Eq. (S7) with codes similar to corresponding ones in ‘*LRparas()*’ function;
  - 3 Calculate LR according to Eq. (S7);
  - 4 Transfer error values, i.e., where no allele shared between P with A or B, into  $(1 - \varphi)$ ;
  - 5 Transfer the LR value into  $\log_{10}$ LR value.
- 

Note that if there is no allele sharing between P with A or B, LR cannot be calculated directly according to Eq. (S7) due to the 0 value of the denominator. In this case,  $(1 - \varphi)$  would be the output.

**Example** Two examples are given, simulating and calculating LR with  $\varphi = 0.5$  for 10,000 A-B-P groups when Hp (A is full-sibling to B) and Hd (NPA is unrelated to NPB) in the formula is true, respectively.

```

1 # Construct pedi data.frames for two types of pedigrees
2 pedi1 <- data.frame(Person=c("F","M","A","B"),
3   Father=c("RI","RI","F","F"),
4   Mother=c("RI","RI","M","M"))
5 pedi2 <- data.frame(Person=c("M","A","B"),
6   Father=c("RI","RI","RI"),
7   Mother=c("RI","M","M"))
8 LR_1=LR_2=data.frame(Log10CLR=rep(0,10000))
9 for (i in 1:42) {
10 # Simulate 10000 groups of A/B/P where A is full sibling of B
11 Genotype1=pedisimu(af = FortytwoSTR$afmatrix[[i ]], ss = 10000,pedi = pedi1)
12 LR_1=LR_1+LRhsip(A=Genotype1[,5:6],B=Genotype1[,7:8],P=Genotype1[,3:4],
13   af = FortytwoSTR$afmatrix[[i ]], rare=FortytwoSTR$rare[i ][1,1])
14 # Simulate 10000 groups of A/B/P where A is half sibling of B, i.e., the true phi=0
15 Genotype2=pedisimu(af = FortytwoSTR$afmatrix[[i ]], ss = 10000,pedi = pedi2)
16 LR_2=LR_2+LRhsip(A=Genotype2[,3:4],B=Genotype2[,5:6],P=Genotype2[,1:2],
17   af = FortytwoSTR$afmatrix[[i ]], rare=FortytwoSTR$rare[i ][1,1])
18 }
19 # histograms of CLR distributions in the two groups
20 xmin<-floor(min(min(LR_1$Log10CLR),min(LR_2$Log10CLR)))
21 xmax<-ceiling(max(max(LR_1$Log10CLR),max(LR_2$Log10CLR)))
22 par(mfrow = c(1, 2))
23 hist (LR_2$Log10CLR,xlab = expression(log[10]~CLR),main = "Fault pedigree",
24   xlim = c(xmin,xmax), col = "red")
25 hist (LR_1$Log10CLR,xlab = expression(log[10]~CLR),main = "True cases",
26   xlim = c(xmin,xmax), col = "blue")

```

### 3.8 ‘*LRgpgcam()*’

The function will compute the likelihood ratio for grandparentage determination using the reference of an uncle or aunt. In such scenarios, a child (C, with genotype *cd*) is claimed to be the grandchild of another individual (GP, with genotype *ef*), and the offspring (A, with genotype *ab*) of the alleged grandparent is involved. Additionally, the genetic information of B’s other parent (M, with genotype *gh*) may or may not be available. Hp assumes that B is the offspring of A’s full sibling, and Hd assumes that B is not related to GP and A. The consideration of inbreeding is not included.

**Derivation of LR in the identification** Similar to the derivation in ‘*LRhsip()*’ function, C can be regarded as individual B in Eq. (21) of the main text and the relationship among other individuals would remain unchanged. Additionally, the genotype of GP and A is unrelated to C’s IBD genotype under Hd. Thus,

$$LR = \frac{\sum_j [\Pr(C \equiv \mathcal{G}_j | A, GP, M, H_p) \times \Pr(\mathcal{G}_j \equiv cd)]}{\sum_k [\Pr(C \equiv \mathcal{G}_k | M, H_d) \times \Pr(\mathcal{G}_k \equiv cd)]} \quad (S8)$$

If label C’s other parent (i.e, GP’s offspring and A’s full-sibling) as NP,  $\Pr(C \equiv \mathcal{G}_j | A, GP, M, H_p)$  can be calculated as follows. Note that the individuals useless in specific reference have been removed from corresponding probabilities.

$$\Pr(C \equiv \mathcal{G}_j | A, GP, M, H_p) = \sum_x [\Pr(NP \equiv \mathcal{G}_x | A, GP, H_p) \times \Pr(C \equiv \mathcal{G}_j | NP \equiv \mathcal{G}_x, M, H_p)] \quad (S9)$$

It is clear that GP, A, and NP can be referred to as P, sibling A, and sibling B respectively in the context of identifying 3.7, and the distribution of NP’s IBD genotype is illustrated in Table S2. Hence, there may be 10 or 5 variations of C’s IBD genotypes, based on the presence or absence of M, and consequently, the LR can be computed as follows if M is available:

$$LR = \frac{d_{GPc}d_{Md} + d_{GPd}d_{Mc}}{4p_c d_{Md} + 4p_d d_{Mc}} + \frac{p_a d_{GPb} (\mathbb{1}_{ac}d_{Md} + \mathbb{1}_{ad}d_{Mc}) + p_b d_{GPa} (\mathbb{1}_{bc}d_{Md} + \mathbb{1}_{bd}d_{Mc})}{4(p_a d_{GPb} + p_b d_{GPa})(p_c d_{Md} + p_d d_{Mc})} + \frac{1}{4} \quad (S10)$$

If M is unavailable,

$$LR = \frac{d_{GPc}}{8p_c} + \frac{d_{GPd}}{8p_d} + \frac{p_a d_{GPb} \mathbb{1}_{ac} + p_b d_{GPa} \mathbb{1}_{bc}}{8p_c (p_a d_{GPb} + p_b d_{GPa})} + \frac{p_a d_{GPb} \mathbb{1}_{ad} + p_b d_{GPa} \mathbb{1}_{bd}}{8p_d (p_a d_{GPb} + p_b d_{GPa})} + \frac{1}{4} \quad (S11)$$

It can be seen that the Eq. (S10) can be transferred into Eq. (S11) by replacing  $d_{Mc}$  and  $d_{Md}$  with  $p_c$  and  $p_d$ , respectively. According to the equation, the function ‘*LRgpgcam()*’ is constructed.

**Input arguments** 7 input arguments are needed for the function

- **A**: A data frame of 2 columns containing the genetic information of individual As;
- **C**: A data frame of 2 columns containing the genetic information of individual Cs;
- **GP**: A data frame of 2 columns containing the genetic information of individual GPs;
- **M**: *NULL* or a data frame of 2 columns containing the genetic information of individual Ms;
- **af**: A data frame similar to **af** in ‘*pairsimu()*’ function;
- **rare**: The frequency of rare alleles, which is similar to **rare** in ‘*LRparas()*’ function;
- **allelename**: The data type of the 4 genotype data frames, similar to **allelename** in ‘*LRparas()*’ function;

**Output value** A data frame of **ss** rows and 1 columns would be output, containing the  $\log_{10}$ LR of each case.

**Detailed description** The function is performed as Algorithm 8.



---

**Algorithm 8: LRgpgcam()**

---

- 1 Conduct formal checks on the 3 genotype data frames containing genotypes, and **stop** if any error exists;
  - 2 Extract the 4  $p$  values in Eq. (S10) with method similar to “#” in ‘*LRparas()*’ function;
  - 3 **if**  $M \neq \text{NULL}$  **then**
  - 4 |   Conduct formal checks on **M** data frame, **stop** if error;
  - 5 |   Calculate  $d_{Mc}$  and  $d_{Md}$ ;
  - 6 **else**
  - 7 |   Set  $d_{Mc} = p_c$  and  $d_{Md} = p_d$ ;
  - 8 **end**
  - 9 Calculate other  $d$  or  $\mathbb{1}$  parameters;
  - 10 Calculate the LR results according to Eq. (S10).
- 

The three parts of Eq. (S10) correspond to the three scenarios of C’s inheritance under Hp: i) C’s paternal allele originated from one of GP’s two alleles; ii) C’s paternal allele is IBD to the allele NP passed to A ; and iii) C’s paternal allele is IBD to the allele NP not passed to A. In the first two components, two factors appear within the denominator:  $(p_a d_{GPb} + p_b d_{GPa})$  and  $(p_c d_{Md} + p_d d_{Mc})$ , which would equal 0 when there is no allele sharing between GP-A and M-C pairs, respectively. Such 0 values would occur due to mutation or the false relationship between the corresponding, and if so, the parts with 0 value in the denominator would be treated as 0 in the LR calculation equation. The output value of LR would have a minimum value of 1/4 because of part iii) in the formula.

**Example** Two examples are given, simulating and calculating CLR based on the 42 STRs in *FortytwoSTR* dataset, for 10,000 A-C-GP-P groups when Hp and Hd in the formula is true, respectively.

```
1 # Construct pedi data.frames for two types of pedigrees
2 pedi1 <- data.frame(Person=c("GF","GM","F","A","M","C"),
3   Father=c("RI","RI","GF","GF","RI","F"),
4   Mother=c("RI","RI","GM","GM","RI","M"))
5 pedi2 <- data.frame(Person=c("GF","GM","F","A","M","C"),
6   Father=c("RI","RI","RI","GF","RI","F"),
7   Mother=c("RI","RI","RI","GM","RI","M"))
8 LR_1=LR_2=data.frame(Log10CLR=rep(0,10000))
9 for (i in 1:42) {
10 # Simulate 10000 group of pedigrees where the Hp is true
11 Genotype <- pedisimu(af = FortytwoSTR$afmatrix[[i ]], ss = 10000,pedi = pedi1)
12 LR_1 <-LR_1+ LRgpgcam(A=Genotype[,7:8],C=Genotype[,11:12],GP=Genotype[,1:2],M=Genotype[,9:10],
13   af=FortytwoSTR$afmatrix[[i ]], rare=FortytwoSTR$rare[i ][1,1])
14 ##Simulate 10000 group of false pedigrees, i.e., P and C is unrelated to GP and A
15 Genotype <- pedisimu(af = FortytwoSTR$afmatrix[[i ]], ss = 10000,pedi = pedi2)
16 LR_2 <-LR_2+ LRgpgcam(A=Genotype[,7:8],C=Genotype[,11:12],GP=Genotype[,1:2],M=Genotype[,9:10],
17   af=FortytwoSTR$afmatrix[[i ]], rare=FortytwoSTR$rare[i ][1,1])
18 }
19 # histograms of CLR distributions in the two groups
20 xmin<-floor(min(min(LR_1$Log10CLR),min(LR_2$Log10CLR)))
21 xmax<-ceiling(max(max(LR_1$Log10CLR),max(LR_2$Log10CLR)))
22 par(mfrow = c(1, 2))
23 hist (LR_2$Log10CLR,xlab = expression(log[10]~CLR),main = "Fault pedigree",
24   xlim = c(xmin,xmax), col = "red")
25 hist (LR_1$Log10CLR,xlab = expression(log[10]~CLR),main = "True cases",
26   xlim = c(xmin,xmax), col = "blue")
```

### 3.9 ‘logLR()’

The function would calculate parameters for a single pairwise case, including CIBS score and multiple types of  $\log_{10}\text{CLR}$ .

**Input arguments** 8 input arguments are needed for the function

---

- **AB**: A data frame of 4 columns containing the genetic information of the two individuals, each two columns for each; Each row of the data frame contains the information of the four allele on a marker, the name of which should be set as the row names. Note that all the row names of this data frame should be included in the data frame names of list **afmatrix** and column names of data frame **rare**, otherwise, the function would report an error;
  - **afmatrix**: *NULL* or a list of multiple data frames containing allele frequency data on multiple markers, which can be generated using 3.1 function. The data frame names should be marker names. For each data frame, there should be 1 column and multiple rows, the column name should be 'Freq' and row names should be allele names on the corresponding marker.
  - **rare**: *NULL* or a data frame of 1 row and multiple columns, containing frequency of rare alleles on each marker. The column names should be the name of each marker;
  - **allelename**: The data type of the genotype data frame, similar to **allelename** in '*LRparas()*' function;
  - **stepPI**: Setting of mutation calculation method similar to **stepwisePI** in '*LRparas()*' function;
  - **adelta3**: Distributions of the IBD coefficient of multiple outbred  $H_p$  in LR calculation, which should be a data.frame with 3 columns and  $x$  rows, where  $x$  denote the number of such LRs being calculated. The names of columns should be '*k0*', '*k1*' and '*k2*', and the names of rows should be the names of each LR. If no outbred LR is needed, set as *NULL*;
  - **adelta9**: Distributions of the IBD coefficient of multiple inbred  $H_p$  in LR calculation, which can be input similar to argument '**adelta3**', and the column names should be '*D1*' to '*D9*'. If no inbred LR is needed, set as *NULL*;
  - **mu**: mutation rate in parentage testing, with a default of 0.002
- 

**Output value** A list of 2 data frames would be output: (i) "results\_on\_each\_marker": a data frame containing calculation results on each marker, with  $nl$  rows and multiple columns, each row for a marker and each column for a type of parameter, including IBS and multiple log10 values of LRs; (ii) "total\_results\_of\_the\_case": a data frame of 1 column and multiple rows, containing the CIBS and log10CLR results for the whole case.

**Detailed description** The function is performed as Algorithm 9.

---

**Algorithm 9: logLR()**

---

- 1 Conduct formal checks on the input arguments, and **stop** if any error exists;
- 2 Calculate IBS score on each marker as the 1st column of “results\_on\_each\_marker” data frame, with method similar to “ $\ddagger$ ” of ‘LRparas()’ function;
- 3 **if** *adelta3=adelta9=NULL* **then**
- 4 |   Output “results\_on\_each\_marker” directly;
- 5 **else**
- 6 |   Extract  $p_c$  and  $p_d$  with a for loop; /\*  $\ddagger$  \*/
- 7 |   Calculate LRid and PInomu, as well as PImu according to **stepPI** setting, similar to ‘LRparas()’ function;
- 8 |   **if** *adelta9=NULL* **then**
- 9 |   |   **for**  $i=1:(\text{row number of } \mathbf{adelta3})$  **do**
- 10 |   |   |   Calculate  $\log_{10}$ LR with LRid, PInomu or PImu, according to  $\kappa$  distribution of the  $i$ th row of **adelta3**;
- 11 |   |   |   Bind the result with “results\_on\_each\_marker” data frame as the  $(i+1)$  column;
- 12 |   |   **end**
- 13 |   **end**
- 14 |   **else**
- 15 |   |   Extract or calculate  $p_a$ ,  $\mathbb{1}_{ab}$  and  $\mathbb{1}_{cd}$ ;
- 16 |   |   Calculate the factors of  $\Delta_1 \sim \Delta_6$  in Eq. (15) of the main text;
- 17 |   |   **if** *adelta3=NULL* **then**
- 18 |   |   |   **for**  $i=1:(\text{row number of } \mathbf{adelta9})$  **do**
- 19 |   |   |   |   Calculate  $\log_{10}$ LR with the LRid, PInomu or PImu, and the 6 factors in line 15, according to  $\Delta$  distribution of the  $i$ th row of **adelta9**;
- 20 |   |   |   |   Bind the result with “results\_on\_each\_marker” data frame as the  $(i+1)$  column;
- 21 |   |   |   **end**
- 22 |   |   |   **else**
- 23 |   |   |   |   Calculate outbred and inbred  $\log_{10}$ LRs one type by one type with two for loops;
- 24 |   |   |   **end**
- 25 |   |   **end**
- 26 |   **end**
- 26 Construct ”total\_results\_of\_the\_case” data frame, by calculate total results of each parameters in “results\_on\_each\_marker” data frame using “as.data.frame(colSums(...))” code;
- 27 Output the 2 data frames “results\_on\_each\_marker” and ”total\_results\_of\_the\_case” as a list.

$\ddagger$  Frequency data would be extracted differently from single locus functions such as ‘LRparas()’, as the frequency data are different among markers. “For()” loop would be needed and the code would differ according to **allelename** setting. Take  $p_c$  as example:

```
1  if (isTRUE(allelename)) {
2    for (i in 1:nl) {
3      para$pc[i]<-afmatrix[[rownames(AB)[i]]][as.character(AB[i,3]),]
4      if (is.na(para$pc[i])){
5        para$pc[i]<-rare[[rownames(AB)[i]]]
6      }
7    }
8  } else {
9    for (i in 1:nl) {
10     para$pc[i]<-afmatrix[[rownames(AB)[i]]][AB[i,3],]
11     if (is.na(para$pc[i])){
12       para$pc[i]<-rare[[rownames(AB)[i]]]
13     }
14   }
15 }
```

**Example** An example is given, simulating and calculating CLR for a pairwise case:

```
1  AB<-data.frame(a=rep(0,42),b=rep(0,42),c=rep(0,42),d=rep(0,42))
2  for (i in 1:42) {
3    temp<-pairsimu(af = FortytwoSTR$afmatrix[[i]], ss = 1, delta = c(0,1,0), allelename = TRUE)
```

```

4 AB[i,]=temp
5 rownames(AB)[i]=names(FortytwoSTR$afmatrix)[i]
6 }
7 adelta3<-data.frame(k0=c(0,0.25,0.5),k1=c(1,0.5,0.5),k2=c(0,0.25,0),row.names=c("PC","FS","HS"))
8 adelta9<-data.frame(D1=0,D2=0,D3=0,D4=0,D5=0.25,D6=0,D7=0.25,D8=0.5,D9=0,row.names="FIMCpair")
9 results<-logLR(AB=AB,afmatrix=FortytwoSTR$afmatrix,rare=FortytwoSTR$rare,stepPI=TURE,adelta3=adelta3,adelta9=adelta9)

```

### 3.10 ‘*testsimulation()*’

An all-in-one simulating and calculating solution for kinship analysis: Combining the genotype generating and LR calculating functions, a integrated function is given for the batch simulation and calculation in kinship analysis, which can generate genotype combinations of multiple individual pairs with specific relationship on multiple independent markers and then calculate multiple types of CLR for each pair.

**Input arguments** 8 input arguments are needed for the function

- **afmatrix**: The allele frequency data utilized in the simulation and calculation, which is in a format similar to the **afmatrix** in the ‘*logLR()*’ function, with the exception that the argument cannot be *NULL*.
- **ss**: The sample size to be simulated;
- **tdelta**: The distribution of  $\Delta$  or  $\kappa$  describing the true relation between the simulated individuals;
- **stepPI**: Setting of mutation calculation method similar to **stepwisePI** in ‘*LRparas()*’ function;
- **adelta3**: Similar to **adelta3** in ‘*logLR()*’ function;
- **adelta9**: Similar to **adelta9** in ‘*logLR()*’ function;
- **pedname**: The name of the simulated relationship;
- **mu**: mutation rate in parentage testing, with a default of 0.002.

**Output value** The output of this function will be a data.frame consisting of *ss* rows and multiple columns. Each row in the data.frame will correspond to the calculation results of a simulated pair defined by the following code. The first column will contain the relation name defined by the argument ‘*pedname*’, while the second column will contain the CIBS value for each pair. The remaining columns will contain the 10 logarithms of multiple types of LRs defined by the arguments ‘*adelta3*’ and ‘*adelta9*’.

**Detailed description** The function will conduct an idealized simulation using allele frequency of multiple markers. Typically, such simulations are useful for the initial assessment of a specific identification panel. Due to its primary nature, some confounding factors are disregarded in the coding of this function, such as excluding mutation factors when generating simulated individuals, which results in rare alleles not being generated and hence, the frequency of rare alleles does not need to be considered. The function is performed as Algorithm 10.

---

#### Algorithm 10: *testsimulation()*

---

- 1 Conduct formal check on the input delta values, and **stop** if any error exists;
  - 2 Extract *nl* as the number of data frames contained in **afmatrix** list;
  - 3 Construct the "result" data frame with *ss* rows and a number of columns determined by the **adelta3** and **adelta9** settings. Set all values in the data frame to 0;
  - 4 **for** *i=1:nl* **do**
  - 5 | Generate genotypes for *ss* individual pairs according to the relationship described by **tdelta**, based on the frequency data contained in the *i*th data frame of the **afmatrix** list, using the ‘*pairsimu()*’ function;
  - 6 | Calculate CIBS and multiple log<sub>10</sub>LR values according to the **adelta3**, **adelta9**, **stepPI**, and **mu** settings, using the ‘*LRparas()*’ function;
  - 7 | Sum the calculation results with the updated "result" data frame;
  - 8 **end**
  - 9 Output the “result” data frame.
-

**Example** An example is given, simulating and calculating 4 types of CLRs for 10,000 parent-child pairs, based on the 42 STRs in *FortytwoSTR* dataset:

```
1  adelta3<-data.frame(k0=c(0,0.25,0.5),k1=c(1,0.5,0.5),k2=c(0,0.25,0),row.names=c("PC","FS","HS"))
2  adelta9<-data.frame(D1=0,D2=0,D3=0,D4=0,D5=0.25,D6=0,D7=0.25,D8=0.5,D9=0,row.names="FIMCpair")
3  data(FortytwoSTR)
4  results<-testsimulation(afmatrix=FortytwoSTR[["afmatrix"]],ss=10000,tdelta=c(0,1,0),adelta3=adelta3,adelta9=adelta9,pedname="PC")
5  results$total_results_of_the_case
```

### 3.11 ‘outputCSV()’

The function would output population data such as “*FortytwoSTR*” into .csv files, which can be used for ‘*EvaluatePanel()*’ function.

**Input arguments** 2 input arguments are needed for the function

- **data**: The name of a list of 4 data frames in format similar to “*FortytwoSTR*”;
- **strpath**: The pathway to output the resulting .csv file;

**Output value** A .csv file in ISFG format, i.e., put allele frequency data in the cells right and down to B2

**Detailed description** The function is performed as Algorithm 11.

#### Algorithm 11: outputCSV()

- 1 Conduct formal check on the input delta values, and **stop** if the data list is not in format similar to *FortytwoSTR*;
- 2 Extract *nl* as the number of data frames contained in ‘afmatrix’ list;
- 3 Extract allele names into a one-column data frame ‘allelenames’ from the row names of the data frames contained in ‘afmatrix’ list;
- 4 Remove the duplicate in ‘allelenames’ data frame and sort it from small to large, set the row number of ‘allelenames’ as *maxa*;
- 5 Construct a data frame “results” with (1+*nl*) columns and (1+*maxa*) rows, set column names as the marker names, data in the 1st column as ‘allelenames’; **for**  $i=1:nl$  **do**
- 6 | Extract the allele frequencies on the *i*th marker into the (*i*+1)th column of ‘results’ data frame and replace the NA values with 0;
- 7 **end**
- 8 Extract the number of individual on each marker into the last row; Outout the result data frame

**Example** An example is given, outputting the *FortytwoSTR* data into a .csv file:

```
1  path<-tempdir()
2  outputCSV(FortytwoSTR,file.path(path,'data.csv'))
```

## 4 Data involved

### 4.1 *FortytwoSTR*

A list of 4 data frames, containing the allele frequency data for the Chinese Han population on a 42-plex STR panel, along with forensic parameters. The list is created using the ‘*EvaluatePanel()*’ function with the default settings, which are: (i) utilizing the values in the last row as the sample size in the population survey; (ii) computing frequencies of rare alleles using the ‘ISFG’ method.

```
1  FortytwoSTR<-EvaluatePanel(strpath = "https://raw.githubusercontent.com/Guanju-Ma/data111/main/42STR.csv")
```

## 4.2 *pediexample*

A data frame containing an example of the input form of **pedi** argument used in *'pedisimu()'* function, which is generated with the following code:

```
1 pediexample<-data.frame(Person=c("GF","GM","F1","F2","M1","M2","A","B"),
2   Father=c("RI","RI","GF","GF","RI","RI","F1","F2"),
3   Mother=c("RI","RI","GM","GM","RI","RI","M1","M2"))
```

which results in a data frame as follows:

	Person	Father	Mother
##1	"GF"	"RI"	"RI"
##2	"GM"	"RI"	"RI"
##3	"F1"	"GF"	"GM"
##4	"F2"	"GF"	"GM"
##5	"M1"	"RI"	"RI"
##6	"M2"	"RI"	"RI"
##7	"A"	"F1"	"M1"
##8	"B"	"F2"	"M2"