Practical Course:
Coalescent Simulations using R.

Sebastian E. Ramos-Onsins.
Centre for Research in Agricultural Genomics (CRAG)

October 12^{th}, 2013

Contents

1 Basic Concepts to Start with R

R is an application used to perform a large number of statistical, mathematical, graphics and other operations that are useful for statistical analysis. The application shows an environment where the user can introduce commands (yes, it's a command line-based application), functions and libraries of functions. The user can also introduce data easily and obtain high quality output plots.
R is a free software. You can download it from http://www.r-project.org. There is a large number of additional functions that can be mainly downloaded from http://cran.r-project.org.
R has an extended help included in the program, although it is not so explicative for beginners. A number of manuals are available for people interested. For example: "Introductory Statistics with R" (Dalgaard, 2002) or "A Handbook of Statistical Analysis Using R" (Everitt and Hothorn, 2006).
When you open R, a command-line window appears. You can write directly any command and the result will be output after pressing "return". You can also open a editor of R scripts to keep all the commands you write (see the Menu). This option is very useful for making functions and save it in a file with ".r" extension. Then, these functions (or a number of commands in a R file) can be used when you need. You simply have to open the file and copy all what you want inside R command-line window.

1.1 Basic commands and operations

1.1.1 Help

The command **help(function)** and **apropos("*any_name*")** are basic functions that allow the user to find and use correctly the commands or functions in R. Using the command **help.start()** you can get access to the complete documentation help menu.

```
> apropos("norm")
[1] "dlnorm" "dnorm" "norm"
[4] "normalise" "normalize_spectrum" "normalizePath"
[7] "plnorm" "pnorm" "qlnorm"
[10] "qnorm" "qqnorm" "qqnorm.default"
[13] "rlnorm" "rnorm" "spnorm"
[16] "sum_spectra_norm"
> help(rnorm)
```

1.1.2 R as a calculator

Using R as a calculator. Sum, divide, square root, power...

```
> 3 + 45
[1] 48
> 48 * 1.2
[1] 57.6
> sqrt(49)
[1] 7
> 8^34
[1] 5.070602e+30
> log(23*exp(4))
[1] 7.135494
```

1.1.3 Keeping the data

It is possible to keep values in variables, not only scalars but also vectors, matrices, or even lists of values.

(i) Keep values in a variables: Example:

```
> x <- 45
> y <- 3.4
> z <- x^2 + 2*y - x/y
> z
[1] 2018.565
```

(ii) Keep data in vectors, matrices or arrays: Using the command **c(*value,values,...*)** is possible to keep values in a vector. Example:

```
> f <- c(1:4)
> f
[1] 1 2 3 4
> g <- c(3,5,6)
> g
[1] 3 5 6
> h <- c(f,g)
> h
[1] 1 2 3 4 3 5 6
```

Exercise 1.1.3.ii: Do an arithmetical sum of two vectors

Using the form **matrix(*values*,nrow=*dim_row*,ncol=*dim_col*)** or the generic command **array(*values*,dim=c(*size_dim1,size_dim2,size_dim3,...*))** it is possible to work with large matrices of values. A matrix must have a unique type of variable, that is, the values are numerical or categorical, but not both at the same time (if you include a letter in a matrix with numbers, all the numbers will be treated as characters).
Example:

```
> m <- matrix(,nrow=4,ncol=3)
> m
     [,1] [,2] [,3]
[1,]   NA   NA   NA
[2,]   NA   NA   NA
[3,]   NA   NA   NA
[4,]   NA   NA   NA
> m <- matrix(0,nrow=4,ncol=3)
> m
     [,1] [,2] [,3]
[1,]    0    0    0
[2,]    0    0    0
[3,]    0    0    0
[4,]    0    0    0
> m <- matrix(c(1:12),nrow=4,ncol=3)
> m
     [,1] [,2] [,3]
[1,]    1    5    9
```

```
[2,] 2 6 10
[3,] 3 7 11
[4,] 4 8 12
> m[2, 3]
[1] 10
> m <- matrix(c(1:12),nrow=4,ncol=3,byrow=TRUE)
> m
  [, 1] [, 2] [, 3]
[1,] 1 2 3
[2,] 4 5 6
[3,] 7 8 9
[4,] 10 11 12
> m[2, 3] #show the value in row 2, column 3
[1] 6
> r <- array(c(1:12),dim=c(2,3,2))
> r
, , 1

  [, 1] [, 2] [, 3]
[1,] 1 3 5
[2,] 2 4 6

, , 2

  [, 1] [, 2] [, 3]
[1,] 7 9 11
[2,] 8 10 12

> r[2, 3, 2]
[1] 12
> r[2, , 1]
[1] 2 4 6
> r[2, -2, 1]
[1] 2 6
```

(iii) The data.frame and the list. The data frame is a set of values (numerical or categorical) that are included in a unique frame and are grouped in a matrix frame. The main difference with a matrix is the possibility to work with both (numerical and categorical) in the same variable. A data frame must have the same number of rows for each defined column. The command is **data.frame(*name_var=values*,...)**.
Example:

```
> d <- data.frame(x=0,y=1:5,cat=c("type1","type2","type1","type1","type2"))
> d
  x y cat
1 0 1 type1
2 0 2 type2
3 0 3 type1
4 0 4 type1
5 0 5 type2
> d$y
[1] 1 2 3 4 5
> d$cat
[1] type1 type2 type1 type1 type2
```

```
Levels:  type1 type2
> d$x[1]
[1] 0
> d$y
[1] 1 2 3 4 5
> d$y>3
[1] FALSE FALSE FALSE TRUE TRUE
> d$cat[3]
[1] type1
Levels:  type1 type2
> d[2,]
  x y cat
2 0 2 S
```

Exercise 1.1.3.iii: Select those values of "cat" that have a value of "y" smaller than 4.

A **list** is any group of variables that have no restriction. Any kind of varible can be included, no matter if the number of rows are different or the variables are categorical or numerical. It is useful for packaging a number of values and variables with a unique name. Example:

```
> l <- list(r=r,f=f,d=d)
> l
$r
, , 1

     [,1][,2][,3]
[1,] 1 3 5
[2,] 2 4 6

, , 2

     [,1][,2][,3]
[1,] 7 9 11
[2,] 8 10 12

$f
[1] 1 2 3 4
$d
  x y cat
1 0 1 A
2 0 2 S
3 0 3 N
4 0 4 Q
5 0 5 E
```

Finally, you can use the function **class(*variable_name*)** to know what is the the type of the variable you are interested in. Example:

```
> f <- c(1:10)
> g <- f*pi
> class(f)
[1] "integer"
> class(g)
```

```
[1] "numeric"
> h <- c(f,g)
> class(h)
[1] "matrix"
```

1.2 Interesting functions in R

There is a large number of functions in R. Here I only introduce you those functions that I consider useful and interesting for this short practical course.

1.2.1 Commands for general purposes

getwd() setwd() library() source() read.table() fread() write.table() which() cbind() rbind() length() sprintf()

(i) getwd(): Shows the working directory. All files you load or save will be in this directory if it is not specified. Example:

```
> getwd()
```

(ii) setwd(): Changes the working directory to the one you specify. Example:

```
> f <- "/Users/sebas/Desktop/PracticalCoal"
> setwd(f)
```

(iii) library(): This command is necessary to load packages of functions. Example:

```
> library(datasets)
```

(iv) source(): This is useful to load specific command from a file.Example:

```
> source("./practica_funciones.R")
```

(v) write.table(): This is useful to save data or results to a file.

```
Usage:
write.table(x, file = "", append = FALSE, quote = TRUE, sep = " ", eol = "\n",
na = "NA", dec = ".", row.names = TRUE, col.names = TRUE, qmethod = c("escape",
"double"))
```

x is the variable you want to write in the file. *file* is the name of the file, between quotes. With *append* is possible to add more data to the file without erasing the content (T) or not (F). *quote* put or not quotes to the values of the variable. *sep* indicates the spacing character. *eol* indicates the character to include at end of the row in a table. *na* is useful in case Non Available data is included. *row.names* indicates if you want the name of the rows are written in the file.
Example:

```
> library(datasets)
> cars[1 : 5,]
  speed dist
1 4 2
2 4 10
3 7 4
```

```
4 7 22
5 8 16
> write.table(x=cars,file="data_example.txt",append=F,
quote=F,eol="\n",dec=".",row.names=F)
```

(vi) read.table(): Include data from a file.

```
Usage:
read.table(file, header = FALSE, sep = "", quote = "\"'", dec = ".", row.names,
col.names, as.is = !stringsAsFactors, na.strings = "NA", colClasses = NA, nrows
= -1, skip = 0, check.names = TRUE, fill = !blank.lines.skip, strip.white =
FALSE, blank.lines.skip = TRUE, comment.char = "#", allowEscapes = FALSE, flush
= FALSE, stringsAsFactors = default.stringsAsFactors(), fileEncoding = "",
encoding = "unknown")
```

Example:

```
> data.cars <- read.table(file="data_example.txt",header=T)
> data.cars[1 : 5,]
  speed dist
1 4 2
2 4 10
3 7 4
4 7 22
5 8 16
```

(vii) fread(): Read data from a file much faster than read.table. You have to install the library "data.read" to use it.

(viii) which(): Indicates the position of the value that you are looking for. Example:

```
> data.cars$dist > 50
[1] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
[13] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE TRUE TRUE FALSE
[25] FALSE TRUE FALSE FALSE FALSE FALSE FALSE FALSE TRUE TRUE TRUE FALSE
[37] FALSE TRUE FALSE FALSE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE
[49] TRUE TRUE
> which(data.cars$dist > 50)
[1] 22 23 26 33 34 35 38 41 42 43 44 45 46 47 48 49 50
```

(ix) cbind(): Merge the columns of two or more variables in the order specified. Example:

```
> g <- data.cars$speed
> speed2 <- cbind(g,2*g)
> speed2[1 : 5,]
g
[1,] 4 8
[2,] 4 8
[3,] 7 14
[4,] 7 14
[5,] 8 16
```

(x) rbind(): Merge the rows of two or more variables in the order specified.

(xi) length(): Gives the size of a vector. Example:

```
> g <- data.cars$speed
> length(g)
[1,] 50
```

(xii) sprintf(): shows text that can include the values of other variables. Example:

```
> a <- c(1.4,3.4)
> t <- c("variable","answer")
> w <- sprintf("The name of the %s is %f, and the %s is %f",t[1],a[1],t[2],a[2])
> w
```

1.2.2 Some algorithmic, statistical and plotting commands

mean() var() apply() quantile() sample() runif() qexp() plot() hist() pdf()

(i) mean(), var(), sd(): Calculate the mean (var or standard deviation, sd) for the values you include. In case of var, when several columns are included, it also calculates de covariance. An interesting option is the possibility to include the "NA" values (na.rm=F) or exclude from the analysis (na.rm=T). Example:

```
> mean(data.cars$dist,na.rm=T)
[1] 42.98
> var(data.cars$dist,na.rm=T)
[1] 664.0608
> var(data.cars,na.rm=T)
          speed      dist
speed 27.95918 109.9469
dist 109.94694 664.0608
```

(ii) apply(): This command allows to use a function over the rows, the columns or both in a matrix.

```
Usage:
apply(X, MARGIN, FUN, ...)
Arguments
X is an array, including a matrix.
MARGIN is a vector giving the subscripts which the function will be applied
over.  1 indicates rows, 2 indicates columns, c(1,2) indicates rows and columns.
FUN the function to be applied (e.g., sum, var ...)
...  optional arguments to FUN.
```

Example:

```
> apply(data.cars,2,mean)
speed dist
15.40 42.98
```

(iii) sample(): This function pick a number of elements from a vector, with or without replacement. Useful for bootstrap resampling. Example:

```
> # 100 Bernoulli trials
> b <- sample(c(0,1), 100, replace = TRUE)
> mean(b)
[1] 0.49 > var(b)
[1] 0.2524242
```

(iv) runif(), rpois(), rnorm(), rexp(): Give a number of values following a Uniform (Poisson, or other) distribution. Example:

```
> u <- runif(n=1000,min=-2,max=2)
> p <- rpois(n=1000,lambda=10)
> n <- rnorm(n=1000,mean=0,sd=1)
> apply(cbind(u,p,n),2,mean)
      u       p       n
0.04939483 10.07100000 0.01193145
> apply(cbind(u,p,n),2,var)
      u       p       n
1.317368 9.373332 0.948162
```

(v) qpois(), qnorm(), qexp(): Starting from the density distribution, we pick a value from the distribution when the P-value is provided. Example:

```
> t_rate <- 1/1000
> uu <- runif(1)
> uu
[1] 0.7117541
> t1 <- qexp(p=uu,rate=t_rate)
> t1
[1] 1242.9414
```

(vi) plot: Plot elements of variables within a graphical display in MANY different ways. Here we only show a simple 2D plot of one variable versus another. Example:

```
> x <- rnorm(1000)
> y <- 2*x*x + rnorm(1000)
> plot(x,y,xlab="x=Normal",ylab="y=2*x^2+Normal")
```

(vi) pdf: You can send the results to a pdf file. All the plot ouputs will be sent to the defined pdf. the file is finally created with the command **dev.off()**:

```
> pdf(file="Example_plot.pdf",height=6,width=10)
> plot(x,y,xlab="x=Normal",ylab="y=2*x^2+Normal")
> dev.off()
```

(vii) quantile(): Calculate the quantiles for the distribution you include. A vector of probabilities can be specified. Example:

```
> n <- rnorm(1000)
> quantile(n)
      0%       25%       50%       75%       100%
-2.96587439 -0.64760592 0.02209920 0.64540246 2.93180125
> quantile(n,probs=c(0.025,0.975)
      2.5%       97.5%
-1.954497 1.887044
```

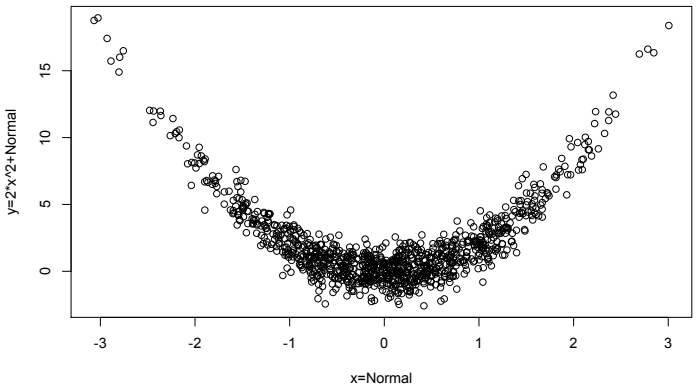

Figure 1: Example of the command plot.

(viii) hist(): Shows the histogram for a vector. Only few options are shown. *n* is the data, *breaks* are the number of segments to divide the data, *freq* shows the absolute number of values in the y-axis (T) or instead the proportion (F). Example:

```
> u <- runif(n=10000,min=-2,max=2)
> p <- rpois(n=10000,lambda=10)
> n <- rnorm(n=10000,mean=0,sd=1)
> par(mfrow=c(1,3))
> hist(u,breaks=30,main="Uniform.  min=-2, max=2",freq=F)
> hist(p,breaks=30,main="Poisson.  lambda=01",freq=F)
> hist(n,breaks=30,main="Normal",freq=F)
```

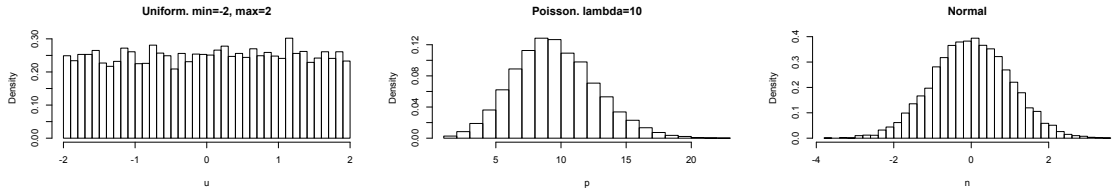

Figure 2: Example of the command hist.

1.2.3 A bit about programming commands

function() for() while() if()

(i) function(): This command is useful to build a list of commands or orders that are interesting to run together (and usually many times). The function can give a number of results at the end. All the commands/orders in the function are within { }. Functions are usually written in a separate file with extension .r and loaded using source("file.r"). Example:

```
#Normalise function
#Function that normalise a value(s) 'x' in relation to a distribution 'y'
```

```
normalise <- function(x,y) {
  result <- (x-(mean(y,na.rm=T)))/sqrt(var(y,na.rm=T))
  result
}

> ns <- rnorm(n=1000,mean=10,sd=23)
> c(mean(ns),sd(ns))
[1] 10.89364 23.63404
> nn <- normalise(ns[1:100],ns)
> c(mean(nn),sd(nn))
[1] -0.1552580 0.9082879
```

(ii) **for()**: Programming a loop. A value is going from the first to the last element that is indicated. Example: we want to calculate the value $a_n = \sum_{i=1}^{n-1} \frac{1}{i}$

```
an <- 0
n <- 20
for(i in 1:(n-1)) {
  an <-an + 1/i
}
an
[1] 3.54774
```

(iii) **while()**: A conditional loop. this loop is only running if the condition inside the parenthesis is true. Example: we will calculate the same value a_n

```
an <- 0
n <- 20
i <- 1
while(i < n) {
  an <- an + 1/i
  i <- i + 1
}
an
[1] 3.54774
```

(iv) **if()/else**: Conditional function. If the condition inside parenthesis is true, run the next line, otherwise skip this line (or section within { }) and go to *else* { } (if defined). Example:

```
tcoal <- 0
if(runif(1) <= 0.95) {
  tcoal <- 1
}
else {
  tcoal <- -1
}
tcoal
[1] 1
```

2 Modeling the Evolutionary Process by using the Coalescent Theory

The fundamentals for Coalescent theory are explained in many reviews published in the last two decades. One of the most interesting, although old, is the one that appear in: Oxford surveys in Evolutionary Biology (Hudson, 1990). You can download it from the R. Hudson web page at http://home.uchicago.edu/~rhudson1/.

Here, the intention is to explain how evolution process can be programmed *in silico* using coalescent fundamentals. In this course, one more file is provided: "ShortCourseCoal_Functions_and_Exercises.R". This file includes all the functions, exercises and solutions that are included here in the guide. Do not try to copy directly the function from the PDF file (it doesn't work). Open the R file and follow the guide of the course using the R console.

2.1 Calculate the time to the ancestor with n = 2 and $N_e = 1000$

Imagine a sample of two individuals and 1000 total (haploid) individuals. The probability that these two individuals have the same parent one generation ago is 1/(1000). We can calculate the number of generations until having the same parent. This is a stochastic process, which means that the result is different each run. The function is:

```
time_coal2_generations <- function(Ne) {
  tg <- 0
  coalescence <- 0
  pcoal <- 1/(Ne)
  while(coalescence == 0) {
    tg <- tg + 1
    if(runif(1) <= pcoal) coalescence <- 1
  }
  tg
}
```

Exercise 2.1:

- Calculate the time to coalescence. Do 1000 replicates and see the differences in the values.
- What happens if you increase/reduce the N_e?

How you can modify the function to include a sudden change in population size 100 generations ago?

```
time_coal2_generations_sudden <- function(Ne,t=1,f=1) {
  tg <- 0
  coalescence <- 0
  pcoal <- 1/(Ne)
  while(coalescence == 0) {
    tg <- tg + 1
    if(t==tg) pcoal <- 1/(Ne*f)
    if(runif(1) <= pcoal) coalescence <- 1
  }
  tg
}
```

2.2 Calculate the time to the ancestor with n = 20 and $N_e = 1000$

For a sample of 20 individuals, the first coalescence is, as an average, much more early, because the probability of having an ancestor is bigger: $\sim (20 * 19/2)/1000$. The function is:

```
time_coaln_generations <- function(Ne,n) {
  tg <- 0
  coalescence <- 0
  pncoal <- 1
  for(i in 1:(n-1)) pncoal <- pncoal * (1-i/Ne)
  pcoal <- 1 - pncoal
  while(coalescence == 0) {
    tg <- tg + 1
    if(runif(1) <= pcoal) coalescence <- 1
  }
  tg
}
```

This probability across the generations follows a exponential distribution. Then, we can simplify the function and increase the speed for the calculation. The command **chose(n,2)** is used to calculate $\binom{n}{2}$:

```
time_coaln_fast_generations <- function(Ne,n) {
  t_rate <- choose(n,2)/Ne
  t1 <- qexp(p=runif(1),rate=t_rate)
  t1
}
```

2.3 Calculate the time to the ancestor for a large (and unknown) N_e

The normal situation is that the Effective number of individuals is unknown, so we are unable to include this value in the simulations. We also assume that the population is large. Also, the sample must be much smaller than the population size. Then, the number of generations calculated is a function of N_e:

```
time_coaln_fast_Ne <- function(n) {
  t_rate <- choose(n,2)/1
  t1 <- qexp(p=runif(1),rate=t_rate)
  t1
}
```

The function to include a sudden change (for example 10x) in population size $0.1N_e$ generations ago (for example) is:

```
time_coaln_fast_Neft <- function(n,f=1,t=1e6) {
  t_rate <- choose(n,2)/1
  t1 <- qexp(p=runif(1),rate=t_rate)
  if(t1 >= t) {
    t_rate <- choose(n,2)/f
    t1 <- t + qexp(p=runif(1),rate=t_rate)
  }
   t1
}
```

and then run:

```
time_coaln_fast_Neft(n=2,f=0.01,t=0.1)
```

2.4 Simulate a sample from the simplest model: the Standard Neutral Model (SNM)

2.4.1 Calculate the Coalescent times for all samples

We can use the same function than in the previous section (obtain the times of coalescence), and calculate the relationships among lineages (constructing the tree):

```
#We use n=20 samples and a population mutation rate θ = 2Nμ = 10.
n <- 20
theta <- 10

#calculate times from n samples to 2 samples:
time <- rep(0,n) #make a vector with of n zero values
for(i in n:2) {
  time[i-1] <- time_coaln_fast_Ne(i) #keep in "time" the times of coalescence
}
time
```

2.4.2 Calculate the Topology of the Coalescent Tree

```
#calculate relationships between samples:  my particular approach.
#At each coalescent event, we have i-1 remaining samples at the time time[i-1].
#Here we will count all samples at every coalescent process, from n->(n-1)->(n-2)->...->1.
#That sums a total of sum(1:n) nodes.
#initialize the number of the node at each coalescence event time.
init <- rep(1,(n+1))
for(i in 2:(n+1)) init[i] <- init[i-1]+(i-1)
init
#calculate the frequencies:  the vector starts counting from the end.
#each node sums the samples that accumulate since present (will be the frequency).
freqs <- array(0,dim=c(sum(1:n)))
#at present each node contains one sample.
freqs[init[n]:(init[n+1]-1)] <- 1
#at each coalescent event, two samples join randomly (panmixia is assumed).
for(i in (n-1):2) {
  #at event, say j, for a sample of i, pick i-2 samples...
  # ...from the previous coalescent event and keep the values.
  freqs[init[i]:(init[i+1]-2)] <- sample(freqs[init[i+1]:(init[i+1]+i)],i-1,replace=F)
  #the last value of the sample will be the the value which sum is n.
  freqs[init[i+1]-1] <- n - sum(freqs[init[i]:(init[i+1]-2)])
}
freqs[1] <- n
```

Showing all the nodes with the value of the samples pending from each node:

```
#the list of samples per node for each coalescent event is the following:
for(i in n:1) show(freqs[init[i]:(init[i+1]-1)])

[1] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 #This is the present sample
[1] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2
[1] 1 1 1 1 1 1 1 1 1 1 1 1 2 1 1 1 2
[1] 1 1 1 1 1 1 1 1 2 1 2 1 1 1 1 2
[1] 2 1 1 1 2 1 1 1 1 1 1 1 1 1 1 3
[1] 1 2 1 2 1 3 1 1 1 1 1 1 1 1 2
[1] 2 2 1 1 2 1 1 1 1 1 1 1 1 4
[1] 1 1 1 1 2 1 1 1 2 1 1 1 6
[1] 1 6 1 1 2 1 2 1 1 1 1 2
[1] 2 1 2 2 1 1 6 1 1 1 2
[1] 1 1 2 2 1 2 1 1 1 8
[1] 2 1 2 1 8 1 1 1 3
[1] 1 1 2 3 1 2 8 2
[1] 2 8 2 1 2 1 4
[1] 4 1 2 8 2 3
[1] 2 3 8 1 6
[1] 6 1 8 5
[1] 1 8 11
[1] 11 9
[1] 20 #this is the ancestor of all samples after n-1 events of coalescence
#note that all rows sum n=20.
```

2.4.3 Include Mutations

The mutations are random events that occur across the time until the coalescence. The process can be simulated by using a Poisson distribution given the population mutation rate and the length of the branches. Here the number of mutations per coalescent event are calculated given using a Poisson distribution with $\lambda = t\mu = \frac{t}{N_e}\frac{2N_e\mu}{2} = T\theta/2$.

```
#calculate the mutations per branch
muts <- array(0,dim=c(sum(1:n)))
for(i in n:2) {
  muts[init[i]:(init[i]+i-1)] <- rpois(i,lambda=time[i-1]*theta/2)
}
```

2.4.4 Calculate the Frequency Spectrum

```
#calculate the frequency spectrum
freq.spectrum <- array(0,dim=c(n))
for(i in 1:sum(1:n)) {
  freq.spectrum[freqs[i]] <- freq.spectrum[freqs[i]] + muts[i]
}
freq.spectrum
[1] 16 6 1 0 1 1 0 1 4 0 3 0 0 0 0 0 0 0 0 0
#16 sngletons, 6 mutations at frequency 2 in the population...etc.
```

All the code here (Times of coalescence (including a sudden change in Ne) + Topology + Mutations + Frequency Spectrum) is included in a single function that gives as an output the values of the frequency spectrum.

```
calculate_freqspectrum_coalescence_f(n,mutpop,tb=0,sb=1)
```

which is a function that need the number of samples, the mutation population parameter (θ) and the optional values for a sudden change in population size.

2.4.5 Calculate estimates of variability based on the frequency spectrum

Different estimates of $2N_e\mu$ are calculated knowing that a unbiased estimator of $\widehat{\theta} = i\xi_i$, where ξ_i is the number of mutations at frequency i (Fu, 1995; Achaz, 2009). Different combinations of weights (ω_i) can be used, giving more or less importance to different sections of the frequency spectrum distribution (Achaz, 2009):

$$\widehat{\theta_\omega} = \frac{1}{\sum_{i=1}^{n-1} \omega_i} \sum_{i=1}^{n-1} \omega_i i \xi_i \tag{1}$$

```
#Function:  calculate S,Watterson theta, nucleotide diversity and others.
calculate_variability_estim <- function(n,freq.spectrum) {
  thetaw <- 0 #Watterson's theta
  thetat <- 0 #Tajima's theta
  thetah <- 0 #Fay and Wu's theta
  thetal <- 0 #Zeng's theta
  for(i in 1:(n-1)) {
    thetaw <- thetaw + freq.spectrum[i] * i * 1/i
    thetat <- thetat + freq.spectrum[i] * i * (n-i)
    thetah <- thetah + freq.spectrum[i] * i * i
    thetal <- thetal + freq.spectrum[i] * i * 1
  }
  thetaw <- thetaw/sum(1/(1:(n-1)))
  thetat <- thetat/sum(n-(1:(n-1)))
  thetah <- thetah/sum(1:(n-1))
  thetal <- thetal/n
  theta1 <- freq.spectrum[1] #Fu and Li's theta
  S <- sum(freq.spectrum) #S is the number of segregating sites

  l <- c(S=S,thetaW=thetaw,thetaT=thetat,theta1=theta1,thetaH=thetah,thetaL=thetal)
}
stats <- calculate_variability_estim(n=n,freq.spectrum=freq.spectrum)
stats
S         thetaW    thetaT   theta1    thetaH   thetaL
33.000000 9.301697 7.994737 16.000000 4.531579 5.950000
```

Exercise 2.4.5: Use the *function calculate_freqspectrum_coalescence_f()* to calculate the frequency spectrum on a sample of $n = 4$ and $\theta = 10$. Then, calculate variability estimates using the function *calculate_variability_estim()*.

2.5 Simulate 1000 samples in a Contraction Population Scenario

Exercise 2.5:

- Do 1000 replicates for $t = 0.1$, $N_e f = 10$, $\theta = 10$, and keep the values of the statistics in a matrix.
- Calculate the 2.5%, 50% and 97.5% quantiles for each statistic and plot the distributions.
- Are the different estimators having the same values? Are they very different to the value included in the simulations ($\theta = 10$)?
- Plot differences between several theta estimators: for example, [theta.Watt - theta.Taj] is the numerator of Tajima's D statistic.
- Repeat the analysis with strength = 1 (SNM) and strength = 1e-1 (Expansion)

2.6 Approximate Bayesian estimation of the mutation population parameter $\theta = 2N_e\mu$ in a population

One of the fundamental analysis in the study of the variability (but also in other fields) is to understand how the observed data fits in a theoretical model (Model-base analysis). In case studying complex biological systems, the fit of the observed data in a model may be very challenging, given the difficulty in obtaining the exact probabilities of the observed data. The Approximate Bayesian Analysis try to solve this problem, simply fitting the data without calculating the exact probabilities (likelihoods). This method uses summary statistics to see how close are the simulated data from the observed data. There is abundant bibliography in the last years about this approximated methodology (see for example Beaumont et al., 2002; Csilléry et al., 2010).
In the following examples we estimate estimate parameters using the ABC method, although the complexity of the models are clearly reduced.

2.6.1 Coalescent Simulation using prior distributions in parameters

A simple Rejection Algorithm is used to estimate the (unique) parameter of the model. The first step of this estimation method is simulate many replicates of the observed data. The simulation follow a chosen theoretical model (here the SNM). Here the SNM has as a unique parameter: θ. This parameter will have many different values, according to a prior uniform distribution. that mean that different values are used in each of the replicates in the simulation:

```
#We want to estimate the parameter "theta" under SNM:
n <- 20 #sample size
nstats <- 1 #number of statistics to calculate

#define min and max values of the parameter theta for the prior
#we will use a uniform distribution
theta.min <- 1
theta.max <- 100

#run simulation
niter <- 25000
sim.data <- array(dim=c(niter,nstats))
colnames(sim.data)[1:nstats] <- c("theta.Watt")

#pick parameter(s) from the initial distribution (uniform)
theta <- runif(niter,theta.min,theta.max)

for(i in 1:niter) {
  #do coalescent tree and calculate statistics.  Each time the theta value is different
  freq.spectrum <- calculate_freqspectrum_coalescence(n=n,mutpop=theta[i])
```

```
  #calculate_freqspectrum_coalescence is the function from Exercise 2.4.4
  variability <- calculate_variability_estim(n=n,freq.spectrum=freq.spectrum)
  sim.data[i,1:nstats] <- variability[2:(nstats+1)]
  #we keep only the "thetaw" statistic
}
```

2.6.2 ABC: Approximate Bayesian Calculation

Once we have a number of simulated replicates calculated given different values of the parameter θ, let's chose those replicates were the simulated Watterson theta's is closer to observed one:

```
obs.data <- c(thetaW=14.1)
dst <- distance.euclidean(nstats,sim.data,obs.data)

#ABC: tolerance at 1%
tol <- 0.01

#Accept or reject parameters according to the threshold (tolerance)
abstol <- quantile(dst,tol,na.rm=TRUE)
wt1 <- dst < abstol
```

Functions that are used to calculate the Euclidian distance between observed and simulated data:

```
#This function calculates euclidean distance between observed and simulated data
distance.euclidean <- function(nstats,sim.data,obs.data) {
#calculate a distance (normalized euclidean) between observed and simulated statistics:
#first normalize
  scaled.sim.data <- sim.data
  for(k in 1:nstats){
    scaled.sim.data[,k] <- normalise(sim.data[,k],sim.data[,k])
  }
  #normalise observed values
  scaled.obs.data <- obs.data
  for(k in 1:npar){
    scaled.obs.data[k] <- normalise(obs.data[k],sim.data[,k])
  }
  #calculate distance
  sum1 <- array(0,dim=c(length(sim.data[,1])))
  for(k in 1:nstats) {
    sum1 <- sum1 + (scaled.sim.data[,k]-scaled.obs.data[k])^2
  }
  dst <- sqrt(sum1)
  dst
}

#Function to normalize data
normalise <- function(x,y){
  if(var(y,na.rm=T) == 0)return (x - mean(y,na.rm=T))
  (x-(mean(y,na.rm=T)))/sqrt(var(y,na.rm=T))
}
```

2.6.3 Results

The results show how to plot two distributions in the same plot (using **lines()**. Also the quantiles of the posterior distribution (in blue cyan in the Figure 3) are shown.

```
#Show Results
par(mfrow=c(1,1))
prior.plot <- hist(x=theta ,breaks=20,plot=F)
post.plot <- hist(x=theta[wt1],breaks=20,plot=F)
maxy <- max(prior.plot$density,post.plot$density)
xlimP <- c(theta.min,theta.max)
ylimP <- c(0,maxy+maxy*0.05)
mt <- "Prior and Posterior distribution of theta"
plot(post.plot,freq=F,main=mt,xlim=xlimP,ylim=ylimP,col="cyan")
lines(prior.plot,freq=F,lty=3,lwd=1)
mean(theta[wt1])
quantile(theta[wt1],prob=c(0,0.025,0.5,0.975,1))
```

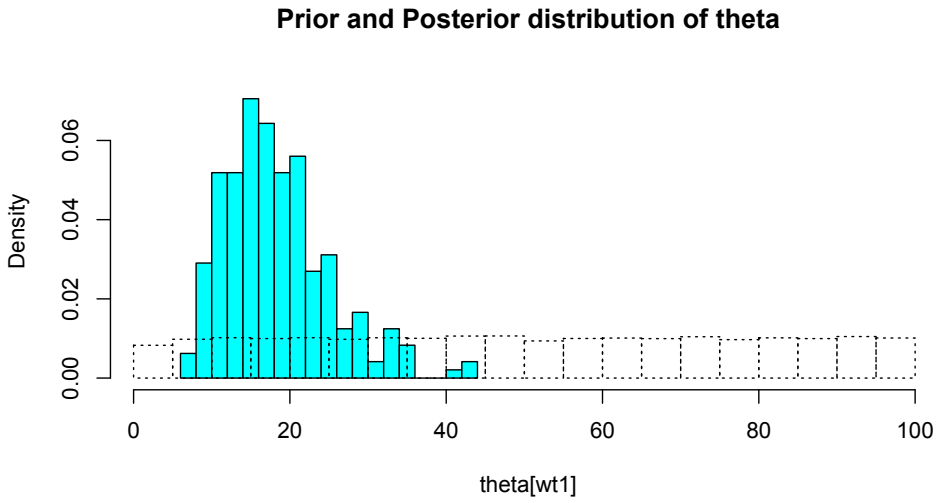

Figure 3: Prior (discontinuous) and Posterior (cyan) Distribution of the parameter θ for the section 2.6.3

2.7 Model Selection: Select the best model (Stationary versus Expansion model) given observed data

Model section refers to a the comparison and selection of two or more models for a better fitting with the observed data. In this section a simple model selection algorithm is explained and results are shown. Other methods of model selection are explained in bibliography (see for example the methods for model selection using ABC in Beaumont, 2008; Fagundes et al., 2007; Toni et al., 2009).

2.7.1 Coalescent Simulations using prior distributions in parameters for the two Models

First we simulate our data under two different models: the SNM (one parameter, θ, and a sudden expansion model with three parameters (θ, T_{exp}, f). Each simulation model is calculated separately, using each parameter a range of values according a prior distribution:

```
#Observed data
n <- 20 #sample size
nstats <- 4 #number of statistics to calculate

#We have two models:  SNM and Expansion
#We simulate the two models separately
time1 <- Sys.time()
niter <- 25000
sim.data.model <- array(dim=c(niter*2,nstats+1))
colnames(sim.data.model)[1:(nstats+1)] <- c("model","theta.Watt","theta.Taj","theta.FL","theta.FW")

#### SNM: 1 parameter:  theta ####
#define min and max values for the prior
theta.snm.min <- 1
theta.snm.max <- 100

#run simulation
sim.data <- array(dim=c(niter,nstats))
#pick parameter(s) from the initial distribution
theta.snm <- runif(niter,theta.snm.min,theta.snm.max)

for(i in 1:niter) {
  #run SNM model and calculate statistics
  freq.spectrum <- calculate_freqspectrum_coalescence(n=n,mutpop=theta.snm[i])
  sim.data[i,1:nstats] <- calculate_variability_estim(n=n,freq.spectrum=freq.spectrum)[2:(nstats+1)]
}
#include the results in a matrix containing the values of the two models
sim.data.model[1:niter,] <- cbind(1,sim.data[1:niter,])

#### Expansion:  3 parameters:  theta,time.exp,strength #####
#define min and max values for the priors
theta.exp.min <- 1
theta.exp.max <- 100
time.exp.min <- 0
time.exp.max <- 1
#change in pop size.  We use a log-uniform distribution
strg.exp.min <- log10(1e-6)
strg.exp.max <- log10(1)

#run simulations
sim.data <- array(dim=c(niter,nstats))
#pick parameter(s) from the initial distribution
theta.exp <- runif(niter,theta.exp.min,theta.exp.max)
time.exp <- runif(niter,time.exp.min,time.exp.max)
strg.exp <- runif(niter,strg.exp.min,strg.exp.max)
strg.exp <- 10^(strg.exp)

for(i in 1:niter) {
  #run exponential model and calculate statistics
  freq.spectrum <- calculate_freqspectrum_coalescence(n=n,mutpop=theta.exp[i],
    tb=time.exp[i],sb=strg.exp[i])
  sim.data[i,1:nstats] <- calculate_variability_estim(n=n,freq.spectrum=freq.spectrum)[2:(nstats+1)]
}
```

```
#include the results in a matrix containing the values of the two models
sim.data.model[(niter+1):(2*niter),] <- cbind(2,sim.data[1:niter,])
```

2.7.2 Contrasting Models using ABC

Model Selection is obtained simply by a Rejection method, where all iterations are included in the same pool and only the closest (1%) distances to the observed data are selected. Then, we count the times the model 1 (SNM) or 2 (Expansion model) are chosen. A large difference, say more than 5 times, is considered a good support in favour to the more frequently selected.

```
##### DO ABC Model Selection:  ########
#Model Selection:
#calculate euclidean distance between observed and simulated data
obs.data <- c(thetaW=14.1,thetatT=9.4,theta1=28,thetaH=5.7)
dst <- distance.euclidean(nstats,sim.data.model[,2:(1+nstats)],obs.data)

#Accept or reject parameters according to the threshold (tolerance)
#ABC: tolerance at 1%
tol <- 0.01
abstol <- quantile(dst,tol,na.rm=TRUE)
wt1 <- dst < abstol

snm <- sum(sim.data.model[wt1,1]==1)
exp <- sum(sim.data.model[wt1,1]==2)
```

2.7.3 Results

Show the results for the selection model. Also, once the model is selected, it is also important to estimate the posterior distribution of the parameters of the model:

```
#Show Results sprintf("Number of SNM values accepted:  %.0f",snm)
sprintf("Number of Exponential values accepted:  %.0f",exp)
sprintf("Ratio Exp/SNM: %.3f",exp/snm)

#wt2 are the choosen values in the Exponential model
wt2 <- sim.data.model[(niter+1):(2*niter),1]==2 & wt1[(niter+1):(2*niter)]
```

Exercise 2.7.3: Show the histograms and the quantiles for the parameter(s) of the selected model. Change the tolerance and compare the results.

3 Inference of Evolutionary parameters and Model Choice Analysis using ABC with *ms* and the R library *abc*

In this exercise we assume that we have a sample of n = 20 and the following estimates of variability:

```
S = 50
Theta(Watterson) = 14.1
Theta(Tajima) = 9.4
Theta(Fay&Wu) = 5.7
```

We want to know wether the Standard Neutral Model (SNM) or a model including a Demographic Expansion (DEM) explains better the observed data.

To do this exercise, we will use the coalescent simulator *ms* (Hudson, 2002) and the R library *abc* (Csilléry et al., 2010) to perform Approximate Bayesian Computation Analysis (Beaumont et al., 2002; Beaumont, 2008).

```
library(abc)
niter <- 50000
OBS <- c(9.4,50,5.7) #Pi,S,H
```

3.1 Simulate statistics under SNM with variable parameter θ

First we need to create a large number (niter=50000) of values of θ given a distribution (here we will use a Uniform between 1 and 100). Go to R:

```
theta.min <- 1
theta.max <- 100
#pick parameter(s) from the initial distribution (uniform)
theta <- runif(niter,theta.min,theta.max)
write.table(x=theta,file="snm_thetap.txt",quote=F,row.name=F,col.name=F)
```

Then, we will simulate 50000 iterations using ms, each iteration will calculate sequences from a coalescent tree with the θ value provided by us. We go to the folder that have the ms program and the file snm_thetap.txt. *ms* needs at least the number of samples, the number of iterations and the value of θ for run. For example:

```
./ms 20 2 -t 10
```

run 2 iterations with a sample of 20 each and with a $\theta = 10$. The output is a scalar giving the number of SNPs, a vector indicating the relative position of each SNP and a matrix with the presence or absence of the derived mutation.
Now we run the simulation using the prior distribution:

```
./ms 20 50000 -t tbs < ./snm_thetap.txt > ./SNM_coaldata.txt
#tbs indicates that this parameter will be provided each time (here from a file)
```

Finally we calculate several statistics using the program sample_stats:

```
./sample_stats < ./SNM_coaldata.txt > ./SNM_statsdata.txt
cut -f 2,4,8,11./SNM_statsdata.txt > ./SNM_statsdata_piSHprior.txt
#we select only the columns for theta(Tajima), S, Theta(Fay&Wu) and the value of
the prior.
```

3.2 Inference of the parameter θ of the SNM using ABC

We will use the R library *abc*. We go to R and do:

```
#read the file:
SNM <- read.table(file="./SNM_statsdata_piSHprior.txt")
head(SNM)
colnames(SNM) <- c("Pi","S","H","Theta.prior")
head(SNM)

#include observed values:
OBS <- c(9.4,50,5.7) #The REAL OBSERVED values.

#run abc:  we wlll use the Beaumont method (local lineal regression)
help(abc)
abc.snm <- abc(target=OBS,param=data.frame(theta=SNM[,4]),
   sumstat=SNM[,-4], tol=0.01,method="loclinear")

#Results:
summary(abc.snm)
hist(abc.snm)
plot(abc.snm,param=data.frame(theta=SNM[,4]))
```

3.3 Validating the parameter estimates

Results can be validated from several ways. Here, the parameter inference can be validated by using random values from the simulation (all simulation values have known population parameter) and calculate the estimation error.

```
snm.val.theta <- cv4abc(param.snm,sumstat=SNM[,-4],nval=100,tols=0.01,method="loclinear")
summary(snm.val.theta)
plot(snm.val.theta)
```

3.4 Parameter inference under the Demographic Expansion Model

In that case, the exponential model has three parameters (theta, time to expansion process, strength of expansion). Remember in coalescent simulations all the events go from present to past. We do the same than before. (i) Define prior distributions for each parameter and calculate niter values; (ii) run coalescent simulations and obtain the summary statistics (iii) infer population parameter using the local lineal regression method from the R function abc; (iv) validate the inference values using a cross validation method.

```
#define min and max values for the priors
theta.exp.min <- 1
theta.exp.max <- 100
time.exp.min <- 0
time.exp.max <- 1
strg.exp.min <- log10(1e-3)
strg.exp.max <- log10(1)

#pick parameter(s) from the initial distribution
theta.exp <- runif(niter,theta.exp.min,theta.exp.max)
time.exp <- runif(niter,time.exp.min,time.exp.max)
```

```
strg.exp <- runif(niter,strg.exp.min,strg.exp.max)
strg.exp <- 10^(strg.exp) #This prior is log10 scaled
write.table(x=cbind(theta.exp,time.exp,strg.exp),file="dem_p.txt",quote=F,row.name=F,col.name=F)

#In command line console:
./ms 20 50000 -t tbs -eN tbs tbs < ./dem_p.txt > ./DEM_coaldata.txt
./sample_stats < ./DEM_coaldata.txt > ./DEM_statsdata.txt
cut -f 2,4,8,11,12,13 ./DEM_statsdata.txt > ./DEM_statsdata_piSHprior.txt

#In R again:
DEM <- read.table(file="DEM_statsdata_piSHprior.txt")
head(DEM)
colnames(DEM) <- c("Pi","S","H","Theta.prior","Time.prior","Strength.prior")
head(DEM)
param.dem <- data.frame(theta=DEM[,4],time=DEM[,5],strength=log10(DEM[,6]))

#Results:
abc.dem <- abc(target=OBS,param=param.dem,sumstat=DEM[,-c(4:6)],tol=0.01,method="loclinear")
summary(abc.dem)
hist(abc.dem)
plot(abc.dem,param=param.dem)
```

3.5 Model Choice by ABC

Model choice use the summary statistics of each model (SNM and DEM) to calculate the distance to the observed values. In that case parameters values are not considered but instead a name for each model is given (for example SNM, DEM) and deal as a categorical value. The best model will be that on that has more values close to the observed one. A local multiple logistic regression can also be used. The validation process is similar to the used with parameters: considering one or another model true from simulations, estimate the error in the model choice.

```
index <- c(rep("SNM",niter),rep("DEM",niter))
sumstat.both <- rbind(SNM[,-4],DEM[,-c(4:6)])
mchoice <- postpr(target= OBS,index=index,sumstat=sumstat.both,tol=0.01,method="rejection")

#Validation
summary(mchoice)
mchoice.val <- cv4postpr(index,sumstat.both,nval=100,tol=0.01,method="rejection")
summary(mchoice.val)
```

4 References

Achaz, G. (2009). Frequency spectrum neutrality tests: One for all and all for one. *Genetics 183*, 249–258.

Beaumont, M. (2008). *Simulations, Genetics and Human Prehistory*, Chapter Joint determination of topology, divergence time and immigration in population trees, pp. 135–15. McDonald Institute Monographs, Univ of Cambridge, Cambridge, UK.

Beaumont, M., W. Zhang, and D. Balding (2002). Approximate bayesian computation in population genetics. *Genetics 162*, 2025–2035.

Csilléry, K., M. Blum, O. Gaggiotti, and O. François (2010). Approximate bayesian computation (abc) in practice. *Trends in Ecology and Evolution 25(7)*, 410–418.

Dalgaard, P. (2002). *Introductory Statistics with R*. Springer.

Everitt, B. and T. Hothorn (2006). *A Handbook of Statistical Analysis Using R*. Chapman and Hall/CRC.

Fagundes, N., N. Ray, M. Beaumont, and S. Neuenschwander (2007). Statistical evaluation of alternative models of human evolution. *PNAS 104(45)*, 17614–16619.

Fu, Y. (1995). Statistical properties of segregating sites. *Theor. Popul. Biol. 48*, 172–197.

Hudson, R. R. (1990). Gene genealogies and the coalescent process. In D. Futuyama and J. Antonovics (Eds.), *Oxford Surveys in Evolutionnary Biology*, Volume 7, pp. 1–45. Oxford: Oxford University Press.

Hudson, R. R. (2002). Generating samples under a wright–fisher neutral model of genetic variation. *Bioinformatics 18(2)*, 337–338.

Toni, T., D. Welch, N. Strelkowa, A. Ipsen, and M. Stumpf (2009). Approximate bayesian computation scheme for parameter inference and model selection in dynamical systems. *Journal of the Royal Society Interface 6*, 187–202.