mmc_R_gendiv: Coalescent simulation and diversity statistics

```
#' Start with a fixed seed to ensure results match the descriptions
seed1 <- 44
set.seed(seed1)
```

Dear user,

this repository contains scripts to replicate our analyses from

- F. Freund, A. Siri-Jégousse "Distinguishing coalescent models - which statistics matter most?", available on biorxiv https://www.biorxiv.org/content/10.1101/679498v2
- F. Menardo, S. Gagneux, F. Freund "Multiple merger genealogies in outbreaks of *Mycobacterium tuberculosis*", bioRxiv doi: <doi: https://doi.org/10.1101/2019.12.21.885723>

It also contains several ready-to-use functions for multiple-merger coalescent simulation and to compute diversity statistics of SNP matrices consisting of zeros and ones (rows=individuals, columns=SNPs), where 0 stands for the wild type and 1 for the mutant type.

We provide a short overview over several functions we deem interesting. We recommend to also have a look at the scripts themselves that we source below, since they consider many more options and additional documentation.

We provide short references to further literature within the text, for the full references please see the two preprints linked above.

We need the following R packages: ape, psych

```
#install.packages("ape","psych")
```

Simulating multiple merger coalescents: Λ-n-coalescents

First, source the coalescent simulator to simulate Λ-n-coalescents.

```
source("general_scripts/lambdacoal_sim.R")
```

This script allows to simulate the jump chain for and to simulate SNP data under any Λ-n-coalescent for which you can provide the transition rates/probabilities of the block counting process. We provide these for Dirac- and Beta(a,b)-n-coalescents. `Lambda_jc_sim` simulates the jump chain of a Λ-n-coalescent. It needs the transition probabilities of the block counting process and the sample size. We simulate the jump chain of a Beta(1,1.5)-n-coalescent.

```
sample_size <- 10
a <- 1
b <- 1.5
beta_jc <- Lambda_jc_sim(sample_size,ratef = function(n){beta_bc_transprob(n,a,b)})
print(beta_jc)
```

```
##      [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10]
## [1,]   -1   -2   -3   -4   -5   -6   -7   -8   -9   -10
## [2,]    2   -2    2   -4   -5    2   -7   -8   -9   -10
## [3,]    3   -2    3   -4    3    3   -7   -8   -9   -10
## [4,]    4   -2    4   -4    4    4   -7   -8   -9    4
## [5,]    4    5    4   -4    4    4    5   -8    5    4
## [6,]    6    5    6   -4    6    6    5    6    5    6
## [7,]    7    7    7   -4    7    7    7    7    7    7
## [8,]    8    8    8    8    8    8    8    8    8    8
```

Rows show the different states of the jump chain until absorption. The equivalence classes of the coalescent after jump $i-1$ are given by the column numbers that share the same entry in row i.

The function `Lambda_seq_sim` simulates data under this jump chain, i.e. given the jump chain, total coalescent rates and a scaled mutation rate θ it produces a 0-1 SNP matrix. The mutation mechanism is a homogeneous Poisson point process on the branches of the Λ-n-coalescent with rate $\frac{\theta}{2}$. The mutations are interpreted under the infinite sites model.

```r
beta_sim1 <- Lambda_seq_sim(beta_jc,theta = 5,
                            totratef = function(n){beta_bc_rate(n,a,b)})
print(beta_sim1)
```

```
##       [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10] [,11] [,12] [,13]
##  [1,]    0    0    0    0    0    0    0    0    1     1     1     0     0
##  [2,]    0    0    0    0    1    0    1    0    0     0     0     1     0
##  [3,]    1    0    0    0    0    0    0    0    1     1     1     0     0
##  [4,]    0    0    0    0    0    0    0    1    0     0     0     0     1
##  [5,]    0    0    0    0    0    0    0    0    1     1     1     0     0
##  [6,]    0    0    0    0    0    0    0    0    1     1     1     0     0
##  [7,]    0    1    1    0    0    1    0    0    0     0     0     1     0
##  [8,]    0    0    0    0    0    0    0    0    1     1     1     0     0
##  [9,]    0    0    0    1    0    0    0    0    0     0     0     1     0
## [10,]    0    0    0    0    0    0    0    0    1     1     1     0     0
##       [,14] [,15] [,16] [,17] [,18]
##  [1,]     1     1     0     0     0
##  [2,]     1     1     0     0     0
##  [3,]     1     1     0     0     0
##  [4,]     0     0     1     1     1
##  [5,]     1     1     0     0     0
##  [6,]     1     1     0     0     0
##  [7,]     1     1     0     0     0
##  [8,]     1     1     0     0     0
##  [9,]     1     1     0     0     0
## [10,]     1     1     0     0     0
```

As one can see, the SNPs are not randomly ordered, but mutations that appear on the same branch of the genealogy between jump times are placed into adjacent columns.

To choose θ based on data, one option is to use to use the generalized Watterson estimator $2s/E(L_n)$, where s is the number of segregating sites observed and $E(L_n)$ is the expected total length of the Λ-n-coalescent (using Möhle '06, Adv. in Appl. Probab., Eq. 2.3). Use the function `eln_f_lambda` to compute $E(L_n)$. Its arguments are the sample size and s, we choose $s = 100$. Setting $s = 0$ returns $E(L_n)$ instead.

```r
source("general_scripts/elength_lambdaxi.R")
theta_w <- eln_f_lambda(sample_size,ratefun = function(n){beta_bc_rates(n,a,b)},
                        s = 100)
```

We will now check whether this does produce s mutations on average. For convenience, we will now use a convenience function to simulate SNP matrices under the Beta-n-coalescent from the script `lambdacoal_sim.R`. We record the number of segregating sites S for 1000 sequences, which is just the number of columns of the SNP matrices. The mean of S should be close to $s = 100$.

```
nsim <- 250
summary(replicate(nsim,ncol(beta_seq_sim(10,a,b,theta_w))))
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##    4.00   65.25   89.00  100.34  126.00  278.00
```

The same approach for Dirac-n-coalescents (with a Dirac measure in $p = 0.5$) would read, using the function `dirac_seq_sim` to simulate sequences under the Dirac-n-coalescent

```
p <- 0.5
theta_w2 <- eln_f_lambda(sample_size,ratefun = function(n){dirac_bc_rates(n,p)},
                         s = 100)
summary(replicate(nsim,ncol(dirac_seq_sim(sample_size,p,theta_w2))))
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##     8.0    61.0    89.5   100.8   128.0   335.0
```

Simulating Ξ-n-coalescents, coalescents with serial sampling and/or population size changes

We also provide similar code to simulate sequences (and jump chains) using Beta-Ξ-n-coalescents (defined as in Blath et al, TPB, 2016) in the script `betaxicoal_sim.R`. Again, the Watterson estimator can also be computed (this needs the script `elength_lambdaxi.R`). Below, the same example of summarising the number of segregating sites in 250 simulations of the Beta-Ξ-10-coalescent is provided.

It should be noted that the rate functions, at least in our implementation, are costly to compute, so a precomputation of rates may be convenient, see the script `distpaper_res/scripts_sim/sim_m4_af_n100.R` for an example.

```
source("general_scripts/betaxicoal_sim.R")
theta_w_betaxi <- eln_f_xi(sample_size,ratefun =  function(n){beta4xi_rates(n,0.5,1.5)},s=100)
summary(replicate(nsim,ncol(betaxi_seq_sim(sample_size,a,b,theta_w_betaxi))))
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   14.00   63.25   90.00   99.99  122.75  330.00
```

We also provide an alternative function to simulate Λ-n-coalescents (and associated SNP sequences) with serial sampling and/or a time change. For serial sampling, you provide the sampling times as a vector *in coalescent time units* (so this is more of a theoretical tool, we have no direct scale translation between coalescent time units and real time units). A serial sampling time of 1 means that the sequence was sampled 1 unit before the last sampling time in the sample. For time-changing the coalescent, which models variable population size, you need to provide the time change as a function. Assume you want to simulate $(\Pi_{\mathcal{G}(t)})_{t\geq 0}$, where $(\Pi_t)_{t\geq 0}$ is a Λ-n-coalescent and $\mathcal{G} : [0,\infty) \to [0,\infty)$ a time change function as described in (e.g.) Freund, JOMB, 2020. Then you need to provide *the inverse of* $\mathcal{G}$ as an input for the script. Of course, you can also use the script for simulating a Λ-n-coalescent without serial sampling (setting sampling times to `rep(0,sample_size)`) and/or without a time change (providing $\mathcal{G}^{-1}(t) = t$)

We do not provide a direct function to compute the Watterson estimator for serial sampling and/or time changes (for time-changed coalescents, these are available, see Spence, Kamm, Song, Genetics 2016 or Matuszewski at al, Genetics 2018). Of course, one can always use Monte Carlo simulation to approximate, see `MTB_MMC_repo/serial_sampling_priorsim.R` for details.

Below, see an example to compute a Dirac-n-coalescent, seen as the limit of an Eldon-Wakeley modified Moran model with $\gamma = 1.5$ (Eldon and Wakeley, Genetics 2006), but with exponential growth with rate $\rho = 10$ as described in Matuszewski et al (Genetics 2018). This means that $\mathcal{G}^{-1}(t) = \frac{\log(1+\rho\gamma(t))}{\rho\gamma}$, see Eq. (33) in Matuszewski et al.

In contrast to the coalescent simulations above, the coalescent simulator provides both the jump times of the process and the waiting times for the next jump (or a further sampling event in serial sampling). The jump chain is recorded as described above, the times are the waiting times between successive jumps.

To simulate SNP matrices, `Lambda_seq_sim_difft` simulates a Poisson process on the branches of this object.

```r
source("MTB_MMC_repo/serial_sampling_priorsim.R")
#p <- 0.5
dirac_exp_coal <- Lambda_difft_coal_tc(coal_times = rep(0,sample_size),
                                       ratef1 = function(n){dirac_bc_rates(n,p)},
                                       Ginv = function(t){log(1+15*t)/15})
print(dirac_exp_coal)
```

```
## $jc
## $jc[[1]]
##  [1]  -1  -2  -3  -4  -5  -6  -7  -8  -9 -10
##
## $jc[[2]]
##  [1]  -1  -2   2  -4   2  -6   2   2  -9 -10
##
## $jc[[3]]
##  [1]  -1  -2   3  -4   3   3   3   3   3 -10
##
## $jc[[4]]
##  [1]   4  -2   3   4   3   3   3   3   3 -10
##
## $jc[[5]]
##  [1]   4   5   5   4   5   5   5   5   5 -10
##
## $jc[[6]]
##  [1] 6 5 5 6 5 5 5 5 5 6
##
## $jc[[7]]
##  [1] 7 7 7 7 7 7 7 7 7 7
##
##
## $coalt
## [1] 0.080105130 0.090083698 0.019173498 0.009100552 0.013095130 0.032268370
```

```r
seq_dirac_exp <- Lambda_seq_sim_difft(coal1 = dirac_exp_coal,theta = 10)
print(seq_dirac_exp)
```

```
##       [,1] [,2] [,3]
##  [1,]    0    0    0
##  [2,]    0    0    1
##  [3,]    0    0    1
##  [4,]    0    0    0
##  [5,]    0    0    1
##  [6,]    0    0    1
##  [7,]    0    0    1
##  [8,]    0    0    1
##  [9,]    1    0    1
## [10,]    0    1    0
```

Diversity statistics: Minimal observable clade sizes

The script `general_scripts/divstats.R` contains (nearly) all diversity statistics we use in the two papers, two further statistics, Tajima's D and Fay and Wu's H are provided from a different repository of colleagues `https://github.com/sdwfrost/popseq` via sourcing `general_scripts/ext_fun_TajD_FWH.R`. Both script are also further documented, if you want to use our implementation of diversity statistics for 0-1 SNP matrices, we recommend reading this documentation, too.

Here, we provide only a limited number of examples for

- computing our new test statistic, the minimal observable clase size. It records, for each sequence in a sample, the minimum count > 1 among SNPs where this sequence shows the mutation (so where the row corresponding to the sequence shows a 1).
- computing the quantiles .1,.3,.5,.7.,.9 of the allele frequencies, of the Hamming distances, of the minimal observable clade sizes as well as the harmonic mean of the latter.

on the sequences `seq_dirac_exp` just generated

```
source("general_scripts/divstats.R")
```

```
## Loading required package: ape

## Loading required package: psych
```

```
#' Quantiles used
quant_v = c(.1,.3,.5,.7,.9)
#' Vector of minimal clade sizes for sequences 1..n
mocs <- vect_OCi(seq_dirac_exp)
#' extract quantiles, harmonic mean (see also function quant_hm_oc)
stats_mocs <- c("qu"=quantile(mocs,quant_v),"hm"=harmonic.mean(mocs))
print(stats_mocs)
```

```
##    qu.10%    qu.30%    qu.50%    qu.70%    qu.90%         hm
##  7.000000  7.000000  7.000000  7.900000 10.000000  7.692308
```

```
#' Quantiles of the Hamming distances between sequences
hammfun(seq_dirac_exp,quant_v)
```

```
## hammd_qu0.1 hammd_qu0.3 hammd_qu0.5 hammd_qu0.7 hammd_qu0.9
##           0           0           1           1           2
```

```
#' Quantiles of the mutant allele frequencies
allele_freqs(seq_dirac_exp,quant_v)
```

```
## AF_qu0.1 AF_qu0.3 AF_qu0.5 AF_qu0.7 AF_qu0.9
##     0.10     0.10     0.10     0.34     0.58
```