fastsimcoal2

a continuous-time coalescent simulator of genomic diversity under arbitrarily complex evolutionary scenarios

Laurent Excoffier

Computational and Molecular Population Genetics lab
Institute of Ecology and Evolution
University of Berne
Baltzerstrasse 6
3012 Berne
Switzerland

Swiss Institute of Bioinformatics
1015 Lausanne, Switzerland

Manual ver 2.5.2.21, November 2015

1. TABLE OF CONTENTS

2. INTRODUCTION

This manual describes the use of *fastsimcoal2,* a program to generate the neutral genomic molecular diversity in current or ancient samples drawn from a population with a complex demographic history. *fastsimcoal2* is a completely rewritten version of *simcoal2* (Laval and Excoffier 2004), a coalescent simulation program implementing a generation by generation approach while *fastsimcoal2* is based on a much faster continuous time approximation. Despite a completely new coalescent engine, *fastsimcoal2* uses exactly the same input files as simcoal2, and it produces very similar output files.

fastsimcoal2 typically generates many replicates of random outcome of molecular diversity under a user-defined evolutionary scenario. The evolutionary scenario is defined in an input parameter file (extension .par) and the output diversity is written in *arlequin* project files (extension .arp) that can then be processed with *arlequin* or *arlsumstat* (Excoffier and Lischer 2010) to get distributions of various summary statistics. Additional options of *fastsimcoal2* can be specified on the command line (type "*fastsimcoal2 -h*" for help on command line options).

fastsimcoal2 can handle very complex evolutionary scenarios including an arbitrary migration matrix between samples, historical events allowing for population resize, population fusion and fission, admixture events, changes in migration matrix, or changes in population growth rates. The time of sampling can be specified independently for each sample, allowing for serial sampling in the same or in different populations.

Different markers, such as DNA sequences, SNP, STR (microsatellite) or multi-locus allelic data can be generated under a variety of mutation models (e.g. finite- and infinite-site models for DNA sequences, stepwise or generalized stepwise mutation model for STRs data, infinite-allele model for standard multi-allelic data).

fastsimcoal2 can simulate data in genomic regions with arbitrary recombination rates, thus allowing for recombination hotspots of different intensities at any position. *fastsimcoal2* implements a new approximation to the ancestral recombination graph in the form of sequential Markov coalescent allowing it to very quickly generate genetic diversity for >100 Mb genomic segments.

Compiled versions of *fastsimcoal2* for Windows, Linux or Mac Os X are available on http://cmpg.unibe.ch/software/fastsimcoal2

Since *fastsimcoal2* output is meant to be interfaced with *Arlequin* or *arlsumstat*, the reader may also want to get more information on Arlequin on http://cmpg.unibe.ch/software/arlequin35

Since ver 2.1, fastsimcoal2 can be used to estimate demographic parameters from the (joint) SFS, as described in Excoffier et al. (2013)

CITATION

The following citations should be used for *fastsimcoal2*:

- Excoffier, L. and M. Foll. 2011. fastsimcoal: a continuous-time coalescent simulator of genomic diversity under arbitrarily complex evolutionary scenarios. Bioinformatics 27: 1332-1334.
- Excoffier, L., Dupanloup, I., Huerta-Sánchez, E., and M. Foll (2013) Robust demographic inference from genomic and SNP data. PLOS Genetics 9(10):e1003905.

DISCUSSION GROUP

A Google discussion group has been created to discuss any issue related to fastsimcoal2. It is available on https://groups.google.com/forum/#!forum/fastsimcoal

ACKNOWLEDGEMENTS

Many people have contributed to the development and improvement of simcoal and fastsimcoal. Historically, John Novembre has contributed to the early development of simcoal, and Guillaume Laval has written most routines dealing with recombination and the ancestral recombination graph in simcoal2. Mathias Currat has also contributed in improving the earlier coalescent code shared between simcoal2 and SPLATCHE. Thierry Schuepbach contributed to the initial development of the multithreaded version. Some piece of code written by all these persons might still be present in some parts of fsc25, even though most has been rewritten when implementing the fastsimcoal2 continuous time coalescent approach, the SMC' recombination framework or the multithreaded version. fastsimcoal2 has also benefitted from many suggestions from Isabel Alvès, Isabelle Duperret, Matthieu Foll, Stephan Peischl, Vitor Sousa, and Benjamin Peter. The use of *tpl* and *est* file is largely inspired by the corresponding settings files of ABCToolBox developed by Daniel Wegmann, and its syntax also follows developments of Samuel Neuenschwander and Mathias Currat in earlier ABC work. The latest developments in fsc25 have been thoroughly tested and debugged by Vitor Sousa. I would also like to thank all users of fastsimcoal(2) who have been reporting bugs directly by email or by using the fastsimcoal Google group list.

3. CHANGES COMPARED TO SIMCOAL2 AND FASTSIMCOAL(2)

FASTSIMCOAL VS. SIMCOAL2

1. Faster continuous-time coalescent simulations
2. Faster recombination model
3. Serial sampling
4. Generation of DNA sequence data under the infinite-site model
5. Sampling of parameter values from prior distributions
6. Computation of population-specific and joint site frequency spectrum
7. Optional output of all trees under the serial Markov coalescent model of recombination
8. RFLP data cannot be simulated anymore

FASTSIMCOAL2 VS. FASTSIMCOAL (JANUARY 2013)

1. Several bug corrections
2. Optional output of all simulated sites (including monomorphic sites)
3. Optional use of a manual seed for the random number generator (--seed xxx command line option)
4. Simulation of ascertained SNP data
5. Generation of the (joint) site frequency spectrum (SFS) from DNA sequence data
6. Generation of multidimensional (>2D) SFS
7. Ability to estimate demographic parameters from the site frequency spectrum inferred from DNA sequences or ascertained SNP chips
8. -s option now requires an additional number specifying the number of SNPs to output

FASTSIMCOAL2.01 VS. FASTSIMCOAL2 (DECEMBER 2013)

1. Several bug corrections

2. User manual update
3. 64 bit version for windows
4. Consolidated par file reading
5. Outputs par file with ML parameters estimation from SFS at the end of the estimation. They are found in file "<template_file_name>_maxL.par".

FASTSIMCOAL2.5 VS. FASTSIMCOAL2.1 (JULY 2014)

1. The fastsimcoal2 program ver2.5 has been renamed **fsc25** (shorter name is better)
2. Use of a different random number generator (same seed will produce different results than in fastsimcoal21)
3. Code optimization resulting in up to 1-75% speed gain for single threaded version (see benchmark)
4. Multithreading (64 bit only), for more speed gain on a multicore processor desktop machine (see benchmark)
5. Result files for parameter estimation now output in separate result directory
6. More options to generate SNP data (see below)
7. New specification for MAF SFS (see below)
8. Added a version for macOSX running in earlier versions (e.g. from 10.6 upwards) (thanks to Iain Mathieson)
9. More tolerant reading of input files (thanks to Allan Strand)
10. Rules in *est files* can now be used for parameter estimations

FASTSIMCOAL2.5.1 VS. FASTSIMCOAL2.5 (SEPTEMBER 2014)

1. Example files are back in zip files (thanks to Alfredo)
2. Description of the exact format of the multiSFS format has been modified in the manual (thanks to Vitor Sousa and Raphael Leblois)
3. Problem in implementing recombination with multiple runs (option -nx where x>1) (thanks to Vitor Sousa and Yang)
4. More precision on branch length when outputing tree in NEXUS format (thanks to Shuo Yang)
5. Faster implementation of recombination under the SMC' algorithm and its extension to multiple recombinations between sites

FASTSIMCOAL2.5.2 VS. FASTSIMCOAL2.5.1 (MARCH 2015)

1. Bug corrections:
   - Fsc251 asked for a joint SFS when two populations samples were listed in tpl file but only one contained active lineages. Bug found by Charleston Chiang.
   - TMRCA was not found in case of recombination and demes with some inactive lineages. Bug found by Ryan Bohlender)
   - Fsc251 was not generating output files when path was provided before input file names (par or tpl).Note that fsc25 should always be run from the directory containing the input files, even though the program can be can be physically located elsewhere. Bug found by Greer Dolby.

- Fsc251 was not taking into account growth rate changes specified in historical events (bug introduced in 2.5.1, and it was not present in ver 2.5.0).

2. -k option has no upper limit anymore, and its default value is 100,000
3. Added new -P command line option, allowing to get the global pooled SFS obtained by pooling all lineages as if in a single population
4. Added two new operators in est file for complex parameters: %min% and %max%
5. Added new functions in est files for complex parameters: abs(), exp(), log(), log10(), pow10()
6. Added a new "bounded" keyword in est file to specify that the upper range of a simple parameter is bounded. Needs to be listed after the "output" or "hide" keywords.
7. Added two new keywords for historical events: "keep" and "nomig".
8. Expected joint SFS is now rescaled such that the sum of sfs entries for polymorphic sites is 1.

FASTSIMCOAL2.5.2.8 VS. FASTSIMCOAL2.5.2 (MAY 2015)

1. Bug corrections:
   - Incorrect simulation of mutations in case of high recombination rates. There was a strong negative correlation between the recombination rate and the number of polymorphic loci, when adjacent sites were the object of recombination. The number of mutation was underestimated for recombination rates, say >1e-7. This bug affected ALL previous fsc releases.
   - Possible overestimation of TMRCA and overall tree size in case of recombination. Bug present since early fsc2 release.
   - Crash of fsc2 in case of very high recombination rate with DNA data
   - Incorrect writing of recombination positions in output arp file when simulating several threads
   - maxObsLhood was not correctly computed when estimation of parameters in a scenario with a single population
   - Change of migration matrix not implemented after first recombination event (thanks to Stefano Mona)
   - Computation of MAF SFS incorrect in case of multiple mutations per site (when -I option not provided and high mutation rates)

2. Speed optimization
3. Output of random DNA nucleotides instead of N for monomorphic loci with the -S option.
4. Possibility to run fsc without command line option if file "fsc_run.txt" is present and contains run path and command line options in current working directory.

FASTSIMCOAL2.5.2.21 VS. FASTSIMCOAL2.5.2.8 (NOVEMBER 2015)

1. Bug corrections:
   - Non implementation of exponential growth at time zero for the first simulated tree. Initial population size therefore does not change for that tree. Note that specifications of exponential growth rates in historical events are correctly implemented even in the first tree. Exponential growth is then correctly implemented in the next simulated trees (thanks to Anand Bhaskar)
   - Crash in case of very large samples sizes (e.g. 60,000) (thanks to Anand Bhaskar)

- Incorrect computation of the max lhood when non integers are used in the observed sfs (thanks to Andi Knautt)
- Reported expected SFS was that of the last iteration and not that associated to the max lhood parameter estimates
- In case of crash due to bad tpl file, parameters reported in file called *<generic name>_bad.par* were not those leading to the crash

2. Speed optimization. Up to 30% speed gain.
3. Output of time to MRCA in file *<generic name>_mrca.txt* with new compiler directive **--recordMRCA**. Beware that this option really slows down computations. Note that we also output the he deme in which MRCA occurred.

4. GETTING STARTED

Compiled version of *fastsimcoal2* and example files can be downloaded from http://cmpg.unibe.ch/software/fastsimcoal2.

The archives include an executable version of *fastsimcoal2* for a given platform, the *fastsimcoal2 pdf* manual, as well as examples *par* files and example template (*tpl*), distribution (*est*), and definition files (*def*) for a variety of simple evolutionary scenarios and several types of markers, with and without recombination. We have also added a few site frequency spectra (SFS) files as well as associated *tpl* and *est* files to try to estimate parameters from the SFS (new to *fastsimcoal2*)

INTALLATION

Unzip the archive file to the directory of your choice. A version of *fastsimcoal2* should be present (*fsc25221.exe* under windows, or *fsc25221* under linux of MacOSX). You can rename it to fsc25 to use the examples found in this manual. On a command line, simply type "`fsc25`" and you will have a list of the different command-line options available to run *fsc25*.

In order to access *fsc25* from any directory on your hard disk, put the directory on your path under windows, or put *fsc25* in your `~/bin` directory under linux or Mac OS X.

RUNNING FASTSIMCOAL2

There are 3 ways to simulate genetic data with *fastsimcoal2*

1) Simulate data under an evolutionary scenario with parameter values defined in an input parameter file

```
fsc25 -i test.par -n 100
```

fastsimcoal2 will use the scenario and the parameter values defined in the parameter file *test.par* and make 100 simulations under this scenario.

2) Simulate data under an evolutionary scenario with parameter values randomly drawn from priors

```
fsc25 -t test.tpl -n 10 -e test.est -E 100
```

fastsimcoal2 will use the scenario defined in the template file *test.tpl* and generate 100 sets of parameter values by randomly drawing these values from the priors defined in the file *test.est*. 10 simulations will be done for each sets of randomly drawn parameter values.

3) Simulate data under an evolutionary scenario with parameter values defined in an external definition file

```
fsc25 -t test.tpl -n 100 -f test.def
```

fastsimcoal2 will use the scenario defined in the template file test.tpl and use the parameter values found in the definition file *test.def*. 100 simulations will be done for each set of predefined parameter values.

Additional descriptions of command line options and input file format can be found in the next chapters

5. STRUCTURE OF INPUT FILES

A SIMPLE UNSUBDIVIDED POPULATION AND DNA SEQUENCES

Let's consider the case of a single population made up of 20000 haploid individuals (or 10000 diploid individuals) where we want to generate diversity along a 10 Kb DNA sequence, with mutation rate $\mu = 2\times10^{-8}$ / bp / gen.

The simcoal2-compatible input file *1popDNA.par* describing such a scenario would look like:

1popDNA.par

```
//Number of population samples (demes)
1
//Population effective sizes (number of genes)
20000
//Sample sizes
10
//Growth rates      : negative growth implies population expansion
0
//Number of migration matrices : 0 implies no migration between demes
0
//historical event: time, source, sink, migrants, new size, growth rate, migr. matrix
0  historical event
//Number of independent loci [chromosome]
1 0
//Per chromosome: Number of linkage blocks
1
//per Block: data type, num loci, rec. rate and mut rate + optional parameters
DNA 10000 0.00000 0.00000002 0.33
```

and with the following command line

```
fsc25 -i 1popDNA.par -n1
```

fastsimcoal2 generates the following *Arlequin*-formatted output file, which is located in a directory having the same name as the input file but with the extension *.arp* removed, *1popDNA* in our case:

./1popDNA/1popDNA_1_1.arp

```
#Arlequin input file written by the simulation program fastsimcoal.exe

[Profile]
      Title="A series of simulated samples"
      NbSamples=1

      GenotypicData=0
      GameticPhase=0
      RecessiveData=0
      DataType=DNA
      LocusSeparator=NONE
      MissingData='?'

[Data]
      [[Samples]]

#Number of independent chromosomes: 1
#Polymorphic positions on chromosome 1
```

```
#288, 800, 1367, 1772, 2045, 2191, 2328, 3258, 3385, 3591, 4162, 5176, 5332, 6128,
6224, 6442, 6455, 7846, 8026, 8536

             SampleName="Sample 1"
             SampleSize=10
             SampleData= {
1_1    1     TAAACATATACCTACGACTC
1_2    1     TAAACATATACCTACGACTC
1_3    1     TTGTACTTTGCCTCCACCTC
1_4    1     TTGTACGTTGCCTCCACCTC
1_5    1     TAAACATATACCTACGACAC
1_6    1     TAAACATATACCTACGACTC
1_7    1     TAGAAATATAACTAAGCATG
1_8    1     GAGTACTTAACGCACACCTC
1_9    1     TTGTACTTTGCCTCCACCTC
1_10   1     TTGTACTTTGCCTCCACCTC

}

[[Structure]]

       StructureName="Simulated data"
       NbGroups=1
       Group={
          "Sample 1"
       }
```

New to *fastsimcoal2,* one now only outputs polymorphic sites for DNA sequences and the position of these sites is provided as a comment above the sample definition.

In order to output all sites, irrespective of their polymorphic status, use the -S command line option.

MIGRATION

fastsimcoal2 can generate data from samples drawn from a subdivided population. For instance the following input file describes an asymmetric 2-deme island model:

2popSTRmigr.par

```
//Number of population samples (demes)
2 samples to simulate :
//Population effective sizes (number of genes)
1000
1000
//Samples sizes
5
5
//Growth rates      : negative growth implies population expansion
0
0
//Number of migration matrices : 0 implies no migration between demes
1
//migration matrix
0.000 0.005
0.001 0.000
//historical event: time, source, sink, migrants, new size, growth rate, migr. matrix
0  historical event
//Number of independent loci [chromosome]
1 0
//Per chromosome: Number of linkage blocks
1
//per Block: data type, num loci, rec. rate and mut rate + optional parameters
MICROSAT  10  0.0000  0.0005 0 0
```

The population sizes are of 1000 genes each, and we want to generate samples of size 5 in each population. One migration matrix is defined, listing the migration rates between the two populations. The migration matrix can be asymmetric, and in the case the entry m_{ij} list the **migration rates backward in time** from population *i* to population *j*. The above-mentioned matrix

```
0.000 0.005
0.001 0.000
```

states that, for each generation **backward in time**, any gene from population 0 has probability 0.005 to be sent to population 1, and that a gene from population 1 has a probability 0.001 to move to population 0.

Note that if no migration matrix is defined, no migration is assumed between populations.

Coming back to our example input file above, the data to be generated consist here of 10 microsatellite markers that are fully linked on a chromosome, with mutation rate $\mu = 5 \times 10^{-4}$ per generation per locus. A pure stepwise mutation model without range constraints is assumed.

The following command line

```
fsc25 -i 2popSTRmigr.par -n1
```

produces the following *Arlequin* output:

2popSTRmigr_1_1.arp

```
#Arlequin input file written by the simulation program fastsimcoal.exe

[Profile]
       Title="A series of simulated samples"
       NbSamples=2

       GenotypicData=0
       GameticPhase=0
       RecessiveData=0
       DataType=MICROSAT
       LocusSeparator=WHITESPACE
       MissingData='?'

[Data]
       [[Samples]]

#Number of independent chromosomes: 1
#Polymorphic positions on chromosome 1
#1, 2, 3, 4, 5, 6, 7, 8, 9, 10

             SampleName="Sample 1"
             SampleSize=5
             SampleData= {
1_1    1      500  499  500  500  499  500  499  500  501  503
1_2    1      498  498  501  502  500  501  498  500  499  499
1_3    1      499  498  500  501  500  501  499  501  502  499
1_4    1      498  498  500  502  500  501  498  501  499  500
1_5    1      498  497  500  503  500  500  498  500  500  499

}
             SampleName="Sample 2"
             SampleSize=5
             SampleData= {
2_1    1      500  499  500  500  499  500  499  500  500  503
2_2    1      498  498  500  501  500  501  498  501  499  499
2_3    1      500  499  500  500  499  500  499  500  500  503
2_4    1      501  499  500  500  499  500  499  500  500  503
2_5    1      499  498  500  501  501  501  499  501  501  499

}

[[Structure]]

       StructureName="Simulated data"
       NbGroups=1
       Group={
          "Sample 1"
          "Sample 2"
       }
```

Note that for microsatellite data, the ancestral allele at each locus is arbitrarily set to have 500 repeats, and that different numbers indicate different number of repeats.

HISTORICAL EVENTS

Historical events can be used to:

- Change the size of a given population
- Change the growth rate of a given population
- Change the migration matrix to be used between population
- Move a fraction of the genes of a given population to another population. This amounts to implementing a (stochastic) admixture or introgression event.
- Move all genes from a population to another population (population). This amounts to fusing two populations into one, looking backward in time, or implementing a population fission looking forward in time.
- One or more of these events at the same time

Note that several events can be defined at different or at the same time in the past, as in the following example file

2popSTRdiv.par

```
//Number of population samples (demes)
2 samples to simulate :
//Population effective sizes (number of genes)
1000
1000
//Samples sizes
5
5
//Growth rates      : negative growth implies population expansion
0
0
//Number of migration matrices : 0 implies no migration between demes
2
//migration matrix
0.000 0.0005
0.0001 0.000
//migration matrix
0.000 0.000
0.000 0.000
//historical event: time, source, sink, migrants, new size, growth rate, migr. matrix
2 historical event
1000 0 0 0 1 0 1
10000 1 0 1 10 0 1
//Number of independent loci [chromosome]
1 0
//Per chromosome: Number of linkage blocks
1
//per Block: data type, num loci, rec. rate and mut rate + optional parameters
MICROSAT  10  0.0000  0.0005 0
```

In this example, we have two migration matrices, and the first migration matrix is used by default until it is changed by a historical event. The second matrix has all entries set to zero, and thus specifies an absence of migrations.

The first historical event

```
1000 0 0 0 1 0 1
```

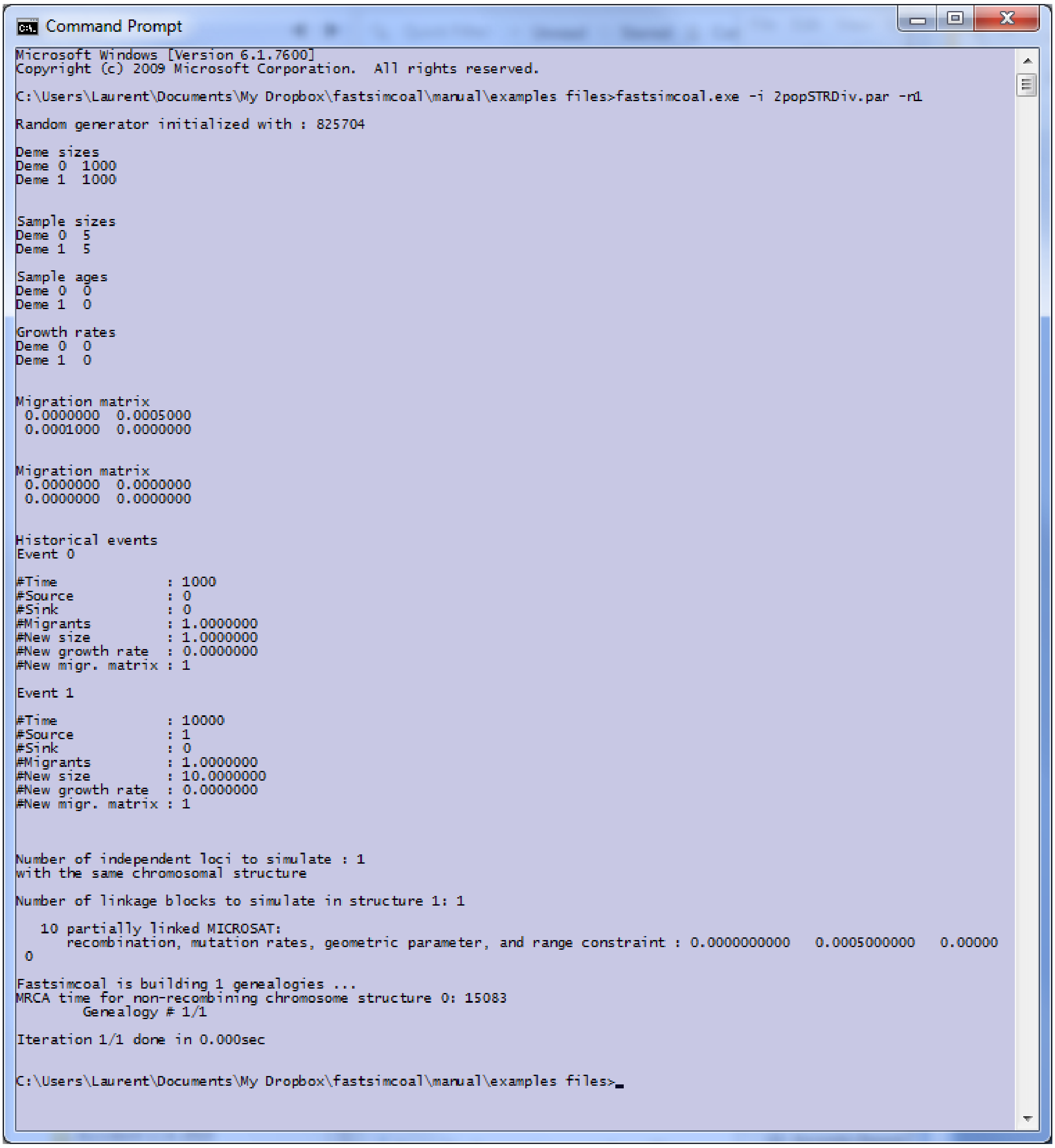
specifies that 1000 generations in the past, we need to change the migration matrix and use migration 1. In doing that it basically stops all migrations between demes.

The second historical event

```
        10000 1 0 1 10 0 1
```

says that 10,000 generations in the past all the genes from deme 1 move to deme 0, the size of which is resized by a factor 10 (to 10,000 genes).

You can check that these historical events are correctly understood by fastsimcoal2 by looking at the console output, which should look like



```
Microsoft Windows [Version 6.1.7600]
Copyright (c) 2009 Microsoft Corporation.  All rights reserved.

C:\Users\Laurent\Documents\My Dropbox\fastsimcoal\manual\examples files>fastsimcoal.exe -i 2popSTRDiv.par -n1

Random generator initialized with : 825704

Deme sizes
Deme 0  1000
Deme 1  1000


Sample sizes
Deme 0  5
Deme 1  5

Sample ages
Deme 0  0
Deme 1  0

Growth rates
Deme 0  0
Deme 1  0


Migration matrix
 0.0000000  0.0005000
 0.0001000  0.0000000


Migration matrix
 0.0000000  0.0000000
 0.0000000  0.0000000


Historical events
Event 0

#Time             : 1000
#Source           : 0
#Sink             : 0
#Migrants         : 1.0000000
#New size         : 1.0000000
#New growth rate  : 0.0000000
#New migr. matrix : 1

Event 1

#Time             : 10000
#Source           : 1
#Sink             : 0
#Migrants         : 1.0000000
#New size         : 10.0000000
#New growth rate  : 0.0000000
#New migr. matrix : 1


Number of independent loci to simulate : 1
with the same chromosomal structure

Number of linkage blocks to simulate in structure 1: 1

   10 partially linked MICROSAT:
      recombination, mutation rates, geometric parameter, and range constraint : 0.0000000000   0.0005000000   0.00000
 0

Fastsimcoal is building 1 genealogies ...
MRCA time for non-recombining chromosome structure 0: 15083
        Genealogy # 1/1

Iteration 1/1 done in 0.000sec


C:\Users\Laurent\Documents\My Dropbox\fastsimcoal\manual\examples files>_
```

Alternatively, the simulations conditions are also output in the file *./2popSTRdiv/2popSTRdiv_1.simparam,* located in the directory *2popSTRdiv*, together with the *Arlequin* project *2popSTRdiv_1_1.arp*.

SERIAL SAMPLING

In *fastsimcoal2,* it is possible to specify at which time sampling was performed in the past. This is simply achieved by adding sampling time after the specification of the sample size. If no time is specified after the sample size, than sampling at present time is assumed, which also ensures compatibility with *simcoal2* input files. The use of this new feature is shown in the simple following input file.

1popDNAserial.par

```
//Number of population samples (demes)
3
//Population effective sizes (number of genes)
1000
0
0
//Sample sizes
5
3 500
2 1000
//Growth rates      : negative growth implies population expansion
0
0
0
//Number of migration matrices : 0 implies no migration between demes
0
//historical event: time, source, sink, migrants, new size, growth rate, migr. matrix
2  historical event
500 1 0 1 1 0 0
1000 2 0 1 1 0 0
//Number of independent loci [chromosome]
1 0
//Per chromosome: Number of linkage blocks
1
//per Block: data type, num loci, rec. rate and mut rate + optional parameters
DNA 10000 0.00000 0.00000002 0.33
```

In this scenario, 5 sequences were sampled at the present time, 3 sequences 500 generations ago, and 2 sequences 1000 generations ago. As we consider that these 3 samples come from the same population, we simply use 2 historical events to transfer these ancient DNA sequences to deme 0 at the time of their sampling.

The resulting genealogy can be visualized by asking *fastsimcoal2* to output coalescent trees. This is done with the *-T* option on the command line. the following command

```
fsc25 -i 1popDNAserial.par -n 2 -T
```

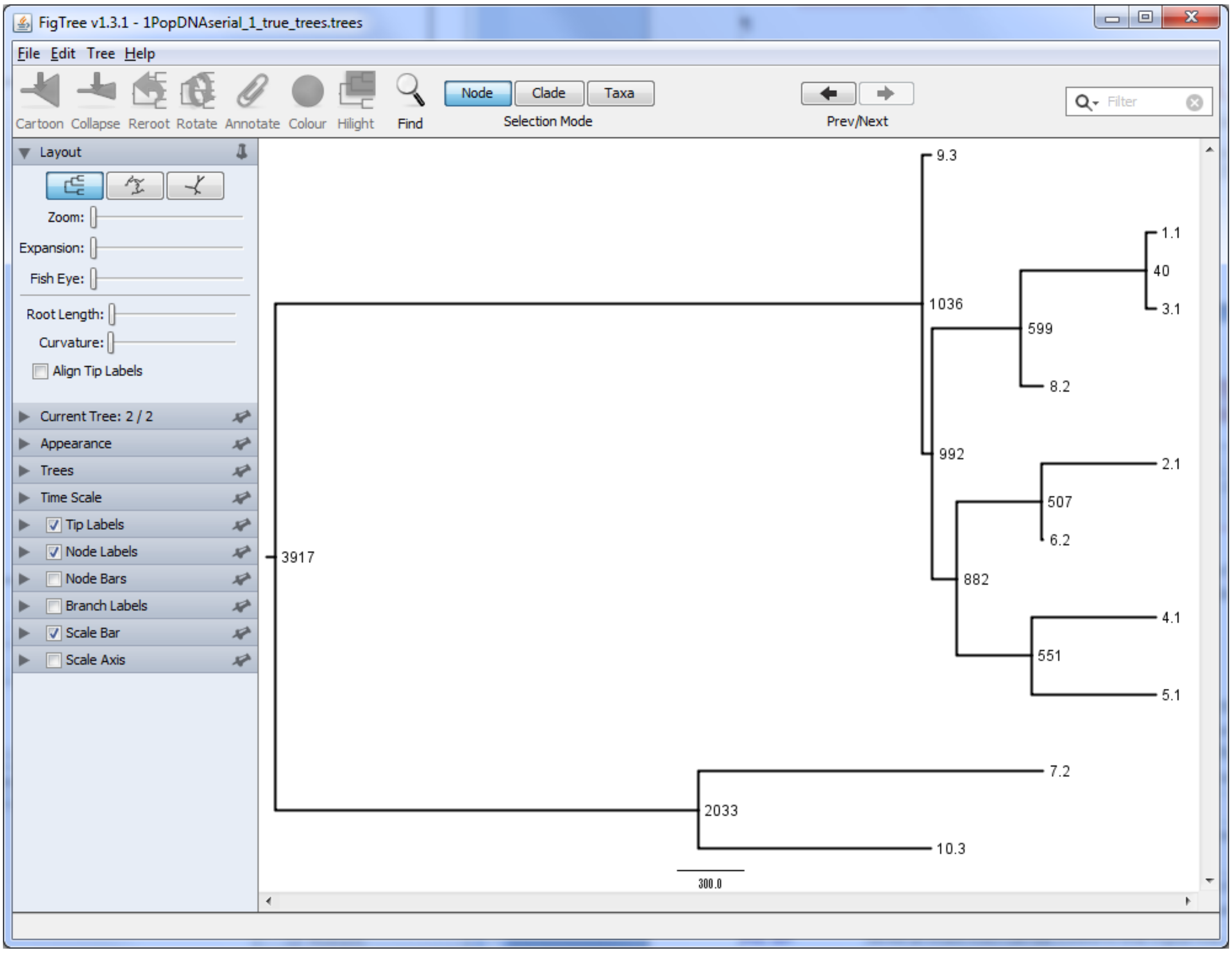
produces tree files in the nexus format, looking like

./1PopDNAserial/1PopDNAserial_1_true_trees.trees

```
#NEXUS
begin trees;  [Treefile generated by fastsimcoal2 .exe (Laurent Excoffier)]

      tree NumGen_tree_1_1_pos_0 = [&U] ((10.3:103, ((4.1:269, 3.1:269):728,
7.2:497):106):1197, ((8.2:580, 9.3:80):852, (6.2:572, (5.1:811, (2.1:55,
1.1:55):756):261):860):368);
      tree NumGen_tree_2_1_pos_0 = [&U] ((9.3:36, (((1.1:40, 3.1:40):559,
8.2:99):393, ((2.1:507, 6.2:7):375, (4.1:551, 5.1:551):331):110):44):2881, (7.2:1533,
10.3:1033):1884);
end;
```

These trees can be conveniently visualized with visualization tools, like *FigTree* (http://tree.bio.ed.ac.uk/software/figtree) freely available for Windows, Linux, or Mac OS X.

our tree of 10 DNA sequences looks like:



We indeed see that the first 5 sequences from deme 0 were sampled at time 0, that the 3 sequences from deme 1 were sampled at time 500, and that the 2 sequences of deme 2 were sampled at time 1000.

Note that it may not be a good idea to record trees for long DNA sequences with recombination, as trees files can become extremely large (>100Mbytes for 10Mbase sequences).

Note also that in fastsimcoal2 branch lengths are now expressed in fractions of generations (e.g. 1205.123)

SIMULATION OF SEVERAL CHROMOSOMAL SEGMENTS

It is easy to simulate several chromosomal segments with different types of markers and different recombination or mutation rates.

In the following example files, one simulates 2 independent non-recombining chromosome segments with the same structure. Each chromosome is made up of 4 blocks, each time with a different mutation model.

1PopMultiLocus.par

```
//Number of population samples (demes)
1
//Population effective sizes (number of genes)
10000
//Sample sizes
5
//Growth rates      : negative growth implies population expansion
0
//Number of migration matrices : 0 implies no migration between demes
0
//historical event: time, source, sink, migrants, new size, new growth rate, migr.
matrix
0  historical event0
//Number of independent loci [chromosome]
2 0
//Per chromosome: Number of linkage blocks
4
//per Block: data type, num loci, rec. rate and mut rate + optional parameters
DNA 1000 0 0.0000002 0.33
SNP 3 0 0
MICROSAT 3 0 0.0005 0 0
STANDARD 2 0 0.001
```

The data section of a typical output of this scenario is :

./1PopMultiLocus/1PopMultiLocus_1_1.arp

```
[Data]
       [[Samples]]

#Number of independent chromosomes: 2
#Polymorphic positions on chromosome 1
#40, 255, 1001, 1002, 1003, 1004, 1005, 1006, 1007, 1008
#Polymorphic positions on chromosome 2
#216, 382, 485, 899, 997, 1001, 1002, 1003, 1004, 1005, 1006, 1007, 1008

             SampleName="Sample 1"
             SampleSize=5
             SampleData= {
1_1    1     CC 000 499  502  499  5  2       AGTGA 100 501  501  503  10  16
1_2    1     CG 011 501  499  500  1  4       CCGCA 011 500  503  498  9   10
1_3    1     GG 000 498  501  500  6  2       ACGCG 001 498  500  498  6   10
1_4    1     GG 100 499  501  500  6  2       ACGCA 001 500  503  499  10  13
1_5    1     CG 000 500  499  501  2  4       ACGCA 001 498  504  498  8   8

}
```

One can also simulate several chromosomes with completely different structures, like

1PopMultiLocusDiffChrom.par

```
//Number of population samples (demes)
1
//Population effective sizes (number of genes)
10000
//Sample sizes
5
//Growth rates      : negative growth implies population expansion
0
//Number of migration matrices : 0 implies no migration between demes
0
//historical event: time, source, sink, migrants, new size, new growth rate, migr.
matrix
0  historical event0
//Number of independent loci [chromosome]
2 1
//Per chromosome: Number of linkage blocks
2
//per Block: data type, num loci, rec. rate and mut rate + optional parameters
DNA 1000 0 0.0000002 0.33
SNP 3 0 0
//Per chromosome: Number of linkage blocks
2
//per Block: data type, num loci, rec. rate and mut rate + optional parameters
MICROSAT 3 0 0.0005 0 0
STANDARD 2 0 0.001
```

The difference with the previous example is that we explicitly tell *fastsimcoal2* to simulate 2 chromosomes with different structures by stating

```
//Number of independent loci [chromosome]
2 1
```

The second number (1) indicates we want to describe different chromosomal structures. Now we need to repeat the block definition for the two structures, while a single block definition was used previously and simply repeated for simulating the two identical chromosomes.

The following output is now produced:

./1PopMultiLocusDiffChrom/1PopMultiLocusDiffChrom_1_1.arp

```
[Data]
       [[Samples]]

#Number of independent chromosomes: 2
#Polymorphic positions on chromosome 1
#85, 105, 169, 277, 372, 470, 629, 635, 702, 934, 960, 998, 1001, 1002, 1003
#Polymorphic positions on chromosome 2
#1, 2, 3, 4, 5

             SampleName="Sample 1"
             SampleSize=5
             SampleData= {
1_1    1     ACCGGCCCACTA 000     502  501  500  19  20
1_2    1     TTATTCGTATTG 111     504  500  501  21  25
1_3    1     ACCGGTCCACTA 000     503  500  501  22  23
1_4    1     TTATGCCTCTCG 111     501  499  506  23  24
1_5    1     TTATGCCTATTG 111     500  498  504  23  21

}
```

RECOMBINATION

Like *simcoal2*, *fastsimcoal2* can generate molecular diversity along recombining chromosomal segments, and recombination rates between adjacent loci are specified just after the definition of the data type and the number of loci of this type, like in the following example file

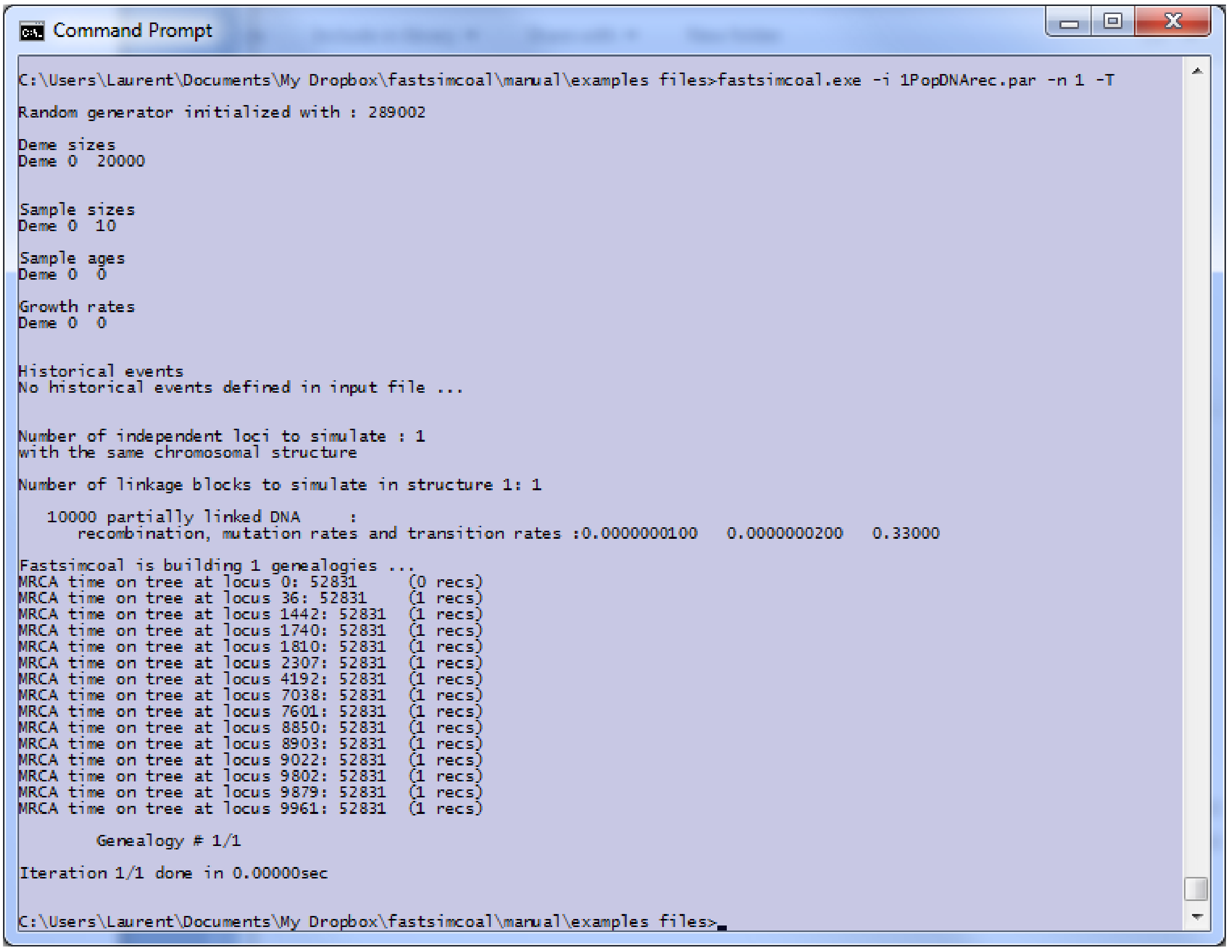
1PopDNArec.par

```
//Number of population samples (demes)
1
//Population effective sizes (number of genes)
20000
//Sample sizes
10
//Growth rates      : negative growth implies population expansion
0
//Number of migration matrices : 0 implies no migration between demes
0
//historical event: time, source, sink, migrants, new size, new growth rate, migr.
matrix
0  historical event
//Number of independent loci [chromosome]
1 0
//Per chromosome: Number of linkage blocks
1
//per Block: data type, num loci, rec. rate and mut rate + optional parameters
DNA 10000 0.00000001 0.00000002 0.33
```

In this case, we want to simulate 10 DNA sequences of 10 Kb with a recombination rate of $r = 10^{-8}$ and mutation rate $\mu = 2 \times 10^{-8}$.

While the syntax to specify recombination is the same as in *simcoal2* , the underlying simulation model is completely different, as we now use a sequential Markov coalescent model (McVean and Cardin 2005) to simulate recombination. In brief, instead of using the classical ancestral recombination graph for the whole chromosomal segment, we generate a first tree on the left of the segment to be simulated. Then we compute where the next recombination event will occur on the segment, implement one recombination event randomly on the tree, detach the recombining lineage from the left tree, and leave this recombining lineage evolve until it coalesces with one of the lineage on the left tree, potentially changing the topology and the height of the tree. See the Sequential Markov Coalescent section for more details.

In the console, *fastsimcoal2* list the position of each recombination breakpoint and the MRCA of the resulting tree, as in



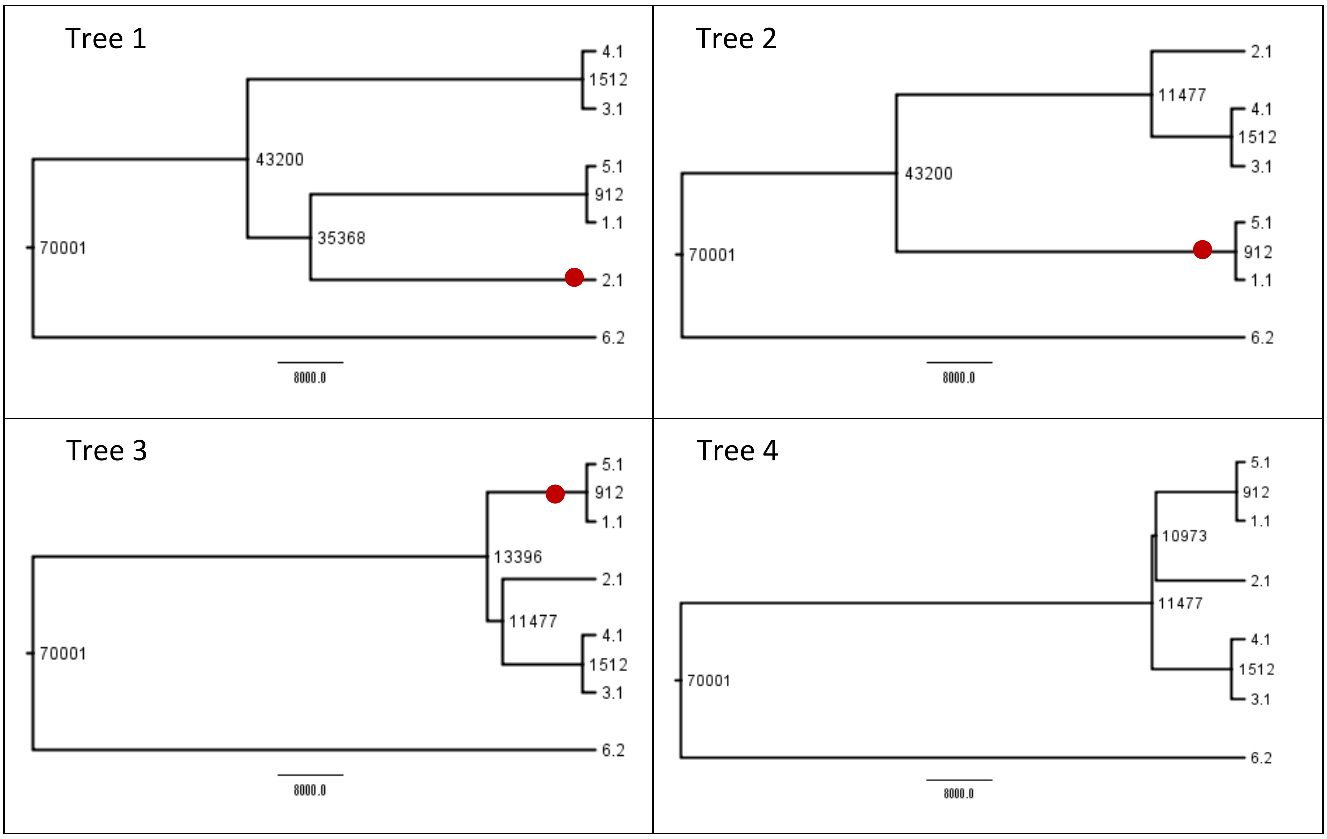
```
C:\Users\Laurent\Documents\My Dropbox\fastsimcoal\manual\examples files>fastsimcoal.exe -i 1PopDNArec.par -n 1 -T

Random generator initialized with : 289002

Deme sizes
Deme 0  20000


Sample sizes
Deme 0  10

Sample ages
Deme 0  0

Growth rates
Deme 0  0


Historical events
No historical events defined in input file ...


Number of independent loci to simulate : 1
with the same chromosomal structure

Number of linkage blocks to simulate in structure 1: 1

   10000 partially linked DNA     :
      recombination, mutation rates and transition rates :0.0000000100   0.0000000200   0.33000

Fastsimcoal is building 1 genealogies ...
MRCA time on tree at locus 0: 52831     (0 recs)
MRCA time on tree at locus 36: 52831    (1 recs)
MRCA time on tree at locus 1442: 52831  (1 recs)
MRCA time on tree at locus 1740: 52831  (1 recs)
MRCA time on tree at locus 1810: 52831  (1 recs)
MRCA time on tree at locus 2307: 52831  (1 recs)
MRCA time on tree at locus 4192: 52831  (1 recs)
MRCA time on tree at locus 7038: 52831  (1 recs)
MRCA time on tree at locus 7601: 52831  (1 recs)
MRCA time on tree at locus 8850: 52831  (1 recs)
MRCA time on tree at locus 8903: 52831  (1 recs)
MRCA time on tree at locus 9022: 52831  (1 recs)
MRCA time on tree at locus 9802: 52831  (1 recs)
MRCA time on tree at locus 9879: 52831  (1 recs)
MRCA time on tree at locus 9961: 52831  (1 recs)

        Genealogy # 1/1

Iteration 1/1 done in 0.00000sec


C:\Users\Laurent\Documents\My Dropbox\fastsimcoal\manual\examples files>_
```

The produced *Arlequin* output file is similar to what would be obtained without recombination, but the nexus tree file now lists all trees produced along the chromosomal segments, as

./1PopDNArec/1PopDNArec_1_true_trees.trees

```
#NEXUS
begin trees;  [Treefile generated by fastsimcoal.exe (Laurent Excoffier)]

	tree NumGen_tree_1_1_pos_0 = [&U] ((5.1:3556, (4.1:2838, 8.1:2838):718):49275, ((2.1:707, (6.1:384, 7.1:384):323):36435, ((3.1:3209, 9.1:3209):12690, (1.1:148, 10.1:148):15751):21243):15689);
	tree NumGen_tree_1_2_pos_36 = [&U] ((5.1:3556, (4.1:2838, 8.1:2838):718):49275, ((2.1:707, (6.1:384, 7.1:384):323):26269, ((3.1:3209, 9.1:3209):12690, (1.1:148, 10.1:148):15751):11077):25855);
	tree NumGen_tree_1_3_pos_1442 = [&U] ((5.1:3556, (4.1:2838, 8.1:2838):718):49275, (((3.1:3209, 9.1:3209):4915, (2.1:707, (6.1:384, 7.1:384):323):7417):18852, (1.1:148, 10.1:148):26828):25855);
	tree NumGen_tree_1_4_pos_1740 = [&U] ((5.1:3556, (4.1:2838, 8.1:2838):718):49275, (((3.1:3209, 9.1:3209):4915, (2.1:707, (6.1:384, 7.1:384):323):7417):25747, (1.1:148, 10.1:148):33723):18960);
...
Etc...
...
end;
```

Note that these trees can also all be browsed individually e.g. in *FigTree* to see the changes in topology produced by the recombination events. For instance, using different settings, here are the changes introduced by 3 recombination events having occurred in the subtree connecting nodes 1 to 5 (approximately located by red dots). Note that additional recombination events fell on other branches of the tree but did not lead to observable changes in tree topology.



INPUT FILE SYNTAX

The syntax of the input file is the same as in *simcoal2*, with a few exceptions.

The input file is divided into the following sections that are each time separated by a comment line. The order of these sections is fixed and cannot be changed.

- Number of populations samples
- Deme sizes
- Sample sizes and sampling times
- Growth rates
- Migration matrices
- Historical events
- Genetic information

NUMBER OF POPULATIONS SAMPLES

This section begins with a comment line. On the second line, the first item should be the number of samples (or demes) to simulate. Additional items on the second line are ignored.

```
//Number of population samples (demes)
1
```

or

```
//Number of population samples (demes)
2 samples to simulate
```

DEME SIZES

This section begins with a comment line, and then has as many lines as demes to be simulated, as mentioned in the previous section.

So if only one sample was defined previously:

```
//Population effective sizes (number of genes)
20000
```

or if, say, three samples were defined previously:

```
//Population effective sizes (number of genes)
20000
10000
100
```

Note that the deme size corresponds here to the number of genes present in a population, which would be the number of individuals for haploid species or to two times the number of individuals for diploid species.

SAMPLES SIZES AND SAMPLING TIMES

This section begins with a comment line, and then lists, for as many samples as defined in the first section, the haploid size of the sample, and optionally the sampling time (the number of generation backward in time when the samples were identified). If the sampling time is omitted, a time of zero is assumed (present sampling).

So the following input

```
//Sample sizes
10
```

is equivalent to

```
//Sample sizes
10 0
```

Other possibilities are:

```
//Sample sizes
10 20
100 1000
```

or

```
//Sample sizes
5
30
0
```

showing in the last case that sample sizes of zero are possible, which just means that no genes were sampled in a given deme at time zero. This additional empty deme may be used in a given evolutionary scenario, for instance as a source population for the currently sampled demes.

GROWTH RATE

This section lists the initial growth rates for all population samples

So the following input

```
//Growth rates     : negative growth implies population expansion
0
```

means that the sampled deme has a stationary population size. Note that growth rates are measured here backward in time, so that negative growth rates imply a forward population expansion. Generally, if the current population size is N_0, and the growth rate is i then the population size *t* generations ago is given by $N_t = N_0 \mathrm{e}^{rt}$.

In the following example, two populations are shrinking backward in time, implying that the sampled demes went through a recent population expansion

```
//Growth rates     : negative growth implies population expansion
-0.001
-0.025
```

Note that the growth rates can be modified at any time by the use of a historical event.

MIGRATION MATRICES

As usual, this section begins with a comment line, and is followed by a line with the number of migration matrices. If there is no migration between demes , then simply enter 0, like

```
//Number of migration matrices : 0 implies no migration between demes
0
```

If we assume that there are two demes connected by migration, then one could enter for instance the following two migration matrices:

```
//Number of migration matrices : 0 implies no migration between demes
2
//migration matrix
0.000 0.0005
0.0001 0.000
//migration matrix
0.000 0.000
0.000 0.000
```

Here the two migration matrices are put below each other separated by a comment line.

The first migration matrix is asymmetric, with deme 0 sending migrant **backward in time** at rate 0.0005 towards deme 1, while deme1 is sending migrants at a lower rate 0.0001 towards deme 0. Therefore, the non-diagonal entries {m_{ij}} of each migration matrix represent the probability for any given lineage to move backward in time from deme *i* to deme *j*. The diagonal entries are ignored. In the above example, the second migration matrix implies an absence of migration. One can switch between migration matrices at any time by means of historical events. Note that by default, the first migration matrix is used at the current time and further back in time until a historical event changes the active migration matrix.

HISTORICAL EVENTS

This section begins with a comment line and is followed by a line specifying the number of historical events. Then, each historical event is defined on a different line. Each historical event is defined by 7 numbers

1) Number of generations *t* before present at which the historical event happened
2) Source deme (the first listed deme has index 0))
3) Sink deme
4) Expected proportion of migrants to move from source to sink. Note that this proportion is not fixed, and that it also represents the probability for each lineage in the source deme to migrate to the sink deme.
5) New size for the **sink** deme, relative to its size at generation *t*.
6) New growth rate for the **sink** deme
7) New migration matrix to be used further back in time.

In the following example, 2 historical events are defined.

```
//historical event: time, source, sink, migrants, new size, growth rate, migr. matrix
2 historical event
1000 0 0 0 1 0 1
10000 1 0 1 10 0 1
```

The first event occurred 1000 generation in the past, and just sets the active migration matrix to matrix 1, whereas matrix 0 had been active until then.

The second event, which occurred 10,000 generations ago, states that all lineages present in deme 1 need to migrate to deme 0. At the same time, the size of deme 1 is increased by a factor 10 and we still use migration matrix 1.

If no historical events are necessary in your simulations just set the number to zero, as

```
//historical event: time, source, sink, migrants, new size, growth rate, migr. matrix
0 historical event
```

Since version 2.5.2, it is possible to specify two additional historical events directives to fastsimcoal:

- *nomig*: if this keyword is added at the end of a historical event definition, migrations between demes are suppressed until the end of the current coalescent simulation. If next in line historical events were to specify the use of some new migration matrix, this would be ignored by fastsimcoal.
- *keep*: if instead of a given value for a **growth rate** or a **migration matrix** one uses the keyword keep, then the former values of these parameters will be used.

These directives are mostly useful in the context of parameter estimation (in *.tpl* files), when the exact timing of a historical event is determined by some parameter, so that the exact sequence of events is not known a priori. Note that `nomig` will prevail over `keep` for the migration matrix if both directives are used for the same historical event.

Example of the use of the `nomig` directive:

```
//historical event: time, source, sink, migrants, new size, growth rate, migr. matrix
2 historical event
1000 0 0 0 1 0 0 nomig  //This directives suppresses migrations between demes
10000 1 0 1 10 0 1      //there will be no migration between demes, even if some are
                          specified in migration matrix 1
```

Example of the use of the *keep* directive:

```
//historical event: time, source, sink, migrants, new size, growth rate, migr. matrix
3 historical event
200 0 0 0 1 -0.001 0
1000 0 0 0 1 keep 1  //Keep current growth rate (-0.001) (and change migration matrix
10000 1 0 1 1 0 keep //Keep migration matrix 1 (plus fuse deme 1 with deme 0, and
                       stop growth rate)
```

GENETIC SETTINGS: CHROMOSOMES, BLOCKS, DATA TYPES, MUTATION, AND RECOMBINATION

GENETIC SETTINGS SUBSECTIONS

The genetic information section has 3 subsections

1) The number of independent **chromosomes** to be simulated and a flag (0, 1) indicating if the different chromosomes have a different (1) or a similar (0) structure. These chromosomes are assumed to be completely unlinked. If the chromosomes have a different structure, the sub-sections 29 and 3) need to be repeated for each chromosome structure.
2) The number of **blocks** to be simulated per chromosome. By block we mean segments of chromosomes that may differ by the type of markers to be simulated, the recombination rate, or the mutation rate. Two consecutive blocks may be of the same type but just differ by the recombination rate, such as for instance simulate a recombination hot spot. Note that the recombination rate between two blocks is that specified in the first block, which is thus valid both within and between blocks.
3) The properties of genetic data to be simulated per block. In each block the following properties need to be specified, in this order:

   i) **Data type**: DNA, MICROSAT, SNP, or STANDARD

   ii) Number of markers with this data type to be simulated. For DNA, this is the sequence length.

   iii) Recombination rate between adjacent markers (between adjacent nucleotides for DNA).

   iv)-vii) Additional data type-specific properties, like the mutation rate per bp and the transition bias for DNA (see list below).

In the example below, we define genetic data to be simulated in two different types of chromosomes:

```
//Number of independent loci [chromosome]
2 1
//Per chromosome: Number of linkage blocks
2
//per Block: data type, num loci, rec. rate and mut rate + optional parameters
DNA 1000 0 0.0000002 0.33
SNP 3 0 0
//Per chromosome: Number of linkage blocks
2
//per Block: data type, num loci, rec. rate and mut rate + optional parameters
MICROSAT 3 0 0.0005 0 0
STANDARD 2 0 0.001
```

On the first type of chromosome we simulate a DNA sequence of 1000 bp and 3 SNPs. On the second type of chromosome, we simulate 3 microsats under a pure stepwise mutation model and 2 multi-allelic loci under an infinite-allele model. On each chromosome, all markers are fully linked as we do not assume any recombination.

In the next example, we want to simulate two chromosomes with the same structure, and we indicate this with the 0 flag after the number of independent chromosomes to simulate.

```
//Number of independent loci [chromosome]
2 0
//Per chromosome: Number of linkage blocks
2
//per Block: data type, num loci, rec. rate and mut rate + optional parameters
DNA 1000 0 0.0000002 0.33
SNP 3 0 0
```

Note that if a sequence of DNA is found monomorphic in a given population sample (due to a too short tree or a too small mutation rate), a question mark `'?'` will be output for each individual sequence.

SPECIFIC PARAMETERS FOR DIFFERENT DATA TYPES

The following optional parameters are required for the different types of markers.

	Extra parameters		
Data types	**4th parameter**	**5th parameter**	**6th parameter**
DNA[1]	Mutation rate per bp	Transition rate (fraction of substitutions that are transitions). A value of 0.33 implies no transition bias.	
MICROSAT	Mutation rate per locus.	Value of the geometric parameter for a Generalized Stepwise Mutation (GSM) model. This value represents the proportion of mutations that will change the allele size by more than one step. Values between 0 and 1 are required. A value of 0 is for a strict Stepwise Mutation Model (SMM).	Range constraint (number of different alleles allowed). A value of 0 means no range constraint
SNP	Minimum frequency for the **derived** allele. Note that if this minimum frequency is not possible given the simulated tree at that locus, the derived allele is coded as a 2 instead of a 1 in the output file.		
STANDARD	Mutation rate per marker. An infinite allele model is assumed.		
FREQ	OUTEXP Output of expected site frequency spectrum for estimated parameters of the model		

[1] Note that by default, *fsc2* uses a finite site model, implying that short DNA sequences simulated with a high mutation rate can be the target of multiple hits. The command-line option `-I` ensures that an infinite-site model is used.

A SPECIAL NOTE ON THE SIMULATION OF SNP DATA

Since ver 2.2.5, we can generate SNP data in different ways, depending on whether we define a mutation rate and whether we use the -I option (infinite site option).

Normally, the mutation rate is irrelevant for generating SNP data, as one assumes that there is a mutation having occurred somewhere on the coalescent tree, but this way of generating SNPs may lead to biased SFS. For instance imagine that we want to generate SNPs in a population that recently went through a bottleneck. In such a population, we expect that many coalescent events will occur during the bottleneck if it is very severe. For some loci, the MRCA might even be located during the bottleneck, and the tree will be very short at those loci. If mutation occur at random, it is very unlikely that they will occur on such short trees. Rather they will occur at loci where the total tree length is large, i.e. when some lineages went through the bottleneck. The relative proportion of the short and long trees will depend on the particular parameter of the bottleneck, but if we ask *fastsimcoal* to generate a SNP irrespective of the length of the tree, the SFS will be biased, as SNPs in short trees will be over-represented.

In order to avoid this situation, we can now specify a mutation rate μ for the SNP. Assuming that at each SNP position mutation occur according to a Poisson process with mean μT, where T is the local total tree length, a SNP will be generated if a randomly drawn Poisson number is larger than or equal to 1. We can thus now have the 4 possible situations:

1. $\mu = 0$: same as *fastsimcoal2*, a mutation is implemented for each SNP, irrespective of total tree length. The resulting SFS will potentially be biased.
2. $\mu > 0$: A SNP will be generated if $x \sim Poisson(\lambda = \mu T) \geq 1$. This should lead to unbiased SFS.
   - If the option -I (infinite site model) is specified on the command line, x SNPs are generated, otherwise, a single SNP is produced if $x \geq 1$.
   - If both options -I and -S are specified on the command line, then even monomorphic sites are output in the arlequin file.

Note that the use of a large mutation rate could be considered as similar to the simulation of a short DNA sequence. For instance if $\mu = 10^{-8}$ per bp, the use of a mutation rate of $\mu = 10^{-6}$ would correspond to the simulation of a short (non-recombining) segment of 100 bp.

EXAMPLES:

1. 1 Mb DNA sequence with recombination rate of 1×10^{-8} and mutation rate of 2×10^{-8}, no transition bias

```
DNA 1000000 1E-8 2E-8 0.33
```

2. 10 Kb DNA sequence without recombination and mutation rate of 2×10^{-6}, high transition bias

```
DNA 10000 0 2E-6 0.95
```

3. 20 MICROSAT with recombination rate of 1×10^{-5} between adjacent markers, a mutation rate of 5×10^{-4} per marker, a geometric GSM parameter of 0.1, and a maximum allele size range of 30.

```
MICROSAT 20 0.00001 5E-4 0.1 30
```

Note that with such a high recombination rate, more than 1 recombination event can occur between adjacent markers, which is adequately taken into account by our modified serial Markov coalescent model (see the description of the SMC' below).

4. 10 SNP sites with variable recombination rates and a minimum derived allele frequency of 0.02.

```
//Number of independent loci [chromosome]
1 0
//Per chromosome: Number of linkage blocks
10
//per Block: data type, num loci, rec. rate + optional parameters
SNP 1 0.00001 0.02
SNP 1 0.001 0.02
SNP 1 0.0002 0.02
SNP 1 0.01 0.02
SNP 1 0 0.02
SNP 1 0.00003 0.02
SNP 1 0.005 0.02
SNP 1 0.00001 0.02
SNP 1 0.002 0.02
SNP 1 0 0.02
```

With this example, one sees how to simulate SNPs located at fixed recombination distances along a given chromosome segment. Like for MICROSAT before, multiple recombination event between adjacent sites will be handled by our SMC' recombination approximation.

5. 20 STANDARD fully linked markers with a mutation rate of 5×10^{-6} per marker.

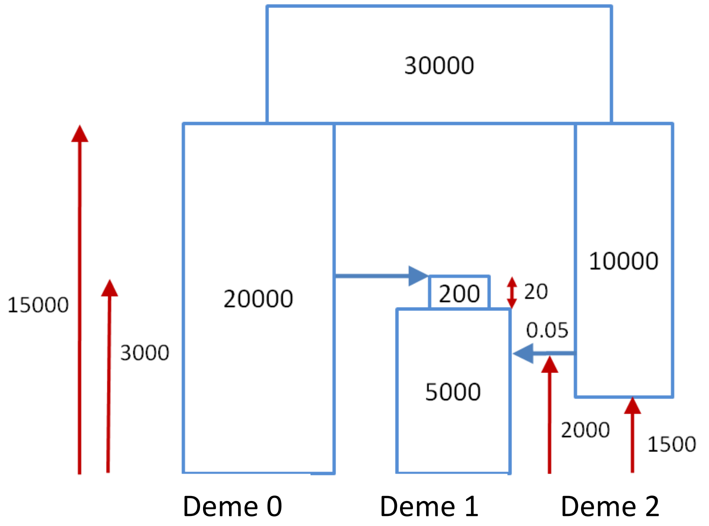
```
STANDARD 20 0 0.000005
```

Note finally that a given chromosome can be simulated for different types of markers, e.g. with a mixture of SNPs and MICROSAT markers:

```
//Number of independent loci [chromosome]
1 0
//Per chromosome: Number of linkage blocks
3
//per Block: data type, num loci, rec. rate + optional parameters
SNP 1 1E-6 0.01
MICROSAT 1 2E-7 5E-4 0 0
SNP 1 0 0.02
```

A RELATIVELY COMPLEX EXAMPLE WITH 3 POPULATIONS, SERIAL SAMPLING, BOTTLENECK, AND INTROGRESSION

Let's assume we are interested in the following relatively complex demographic scenario involving 3 demes (populations).



This scenario assumes that genes from deme 2 were sampled 1500 generations ago, and that 5% of the genes present in deme 1 2000 generations ago actually came from deme 2 (which implies an introgression or admixture event). It also assumes that deme 1 originated from deme 0 2000 ago and went through a bottleneck size of 200 (haploid) individuals during 20 generations before recovering a size of 5000 (haploid) individuals. Finally deme 0 and deme 2 diverged 15000 generations ago from an ancestral population of size 30000, to form tow populations of size 20000 and 10000, respectively.

We can simulate the evolution of 6 10Mb DNA sequences for deme2, and 20 10Mb sequences in both deme 0 and deme 1 under this scenario with the following parameter file *3popDNASFS.par*:

3popDNASFS.par

```
//Number of population samples (demes)
3
//Population effective sizes (number of genes)
20000
5000
10000
//Sample sizes
20
20
6 1500
//Growth rates: negative growth implies population expansion
0
0
0
//Number of migration matrices : 0 implies no migration between demes
0
//historical event: time, source, sink, migrants, new size, new growth rate, migr. matrix
4  historical event
2000 1 2 0.05 1 0 0
2980 1 1 0 0.04 0 0
3000 1 0 1 1 0 0
15000 0 2 1 3 0 0
//Number of independent loci [chromosome]
1 0
//Per chromosome: Number of linkage blocks
1
//per Block: data type, num loci, rec. rate and mut rate + optional parameters
DNA 10000000 0.00000001 0.00000002 0.33
```

6. SAMPLING PARAMETER VALUES FROM PRIOR DISTRIBUTIONS

fastsimcoal2 has a built-in procedure to sample parameters from prior distributions. The principle is very much the same as that implemented in *ABCToolBox* (Wegmann et al. 2010), using a template file (**.tpl*) where parameters to be sampled are input as keyword, and an estimation file (**.est*) where parameter distributions are fully specified. Such a way to simulate data is invoked e.g. with the following command line arguments

```
fsc25 -t 1popDNArand.tpl -n10 -e 1popDNArand.est -E 1000
```

which tells *fastsimcoal2* to use the template file *1popDNArand.tpl* and the estimation *1popDNArand.est* to generate 10 simulations for each of the 1000 sets of randomly drawn parameter values.

TEMPLATE FILE

An example template file reads as follows:

1popDNArand.tpl

```
//Number of population samples (demes)
1
//Population effective sizes (number of genes)
NPOP
//Sample sizes
10
```

```
//Growth rates      : negative growth implies population expansion
0
//Number of migration matrices : 0 implies no migration between demes
0
//historical event: time, source, sink, migrants, new size, new growth rate, migr.
matrix
1  historical event
TEXP 0 0 0 RESIZE 0 0
//Number of independent loci [chromosome]
1 0
//Per chromosome: Number of linkage blocks
1
//per Block: data type, num loci, rec. rate and mut rate + optional parameters
DNA 10000 0.00000 MUTRATE 0.33
```

As can be seen above, a template file has exactly the same structure as a parameter file, but some keywords (NPOP, TEXP, RESIZE, and MUTRATE) are present instead of actual parameter values.

Those keywords will be substituted with actual parameter values by *fastsimcoal2,* if the distributions of these parameters are found in the estimation file.

CAUTION

Keyword for parameter need to be a *unique* string of characters. By unique, we mean that a keyword cannot be part of another keyword. Note also that *fastsimcoal2* is not case sensitive, and therefore NPOP and nPop are identical for *fastsimcoal2*. So, for instance, NPOP and NPOP1 are incompatible as NPOP is included into NPOP1, and if NPOP is defined before NPOP1 in the est file, the value of NPOP (say 100) will also be inserted into NPOP1, resulting of a value of 1001 for NPOP1. Such problems are difficult to spot, and often go un-noticed and result in bad estimations. It has been suggested by some users that parameter keywords could be ended by a “$” sign, which is a good idea. In that case NPOP and NPOP1 would become NPOP$ and NPOP1$, which are now compatible. Any other ending character (other than “\”) would be okay, as well.

ESTIMATION FILE

An estimation file matching the template file *1popDNArand.tpl* would look as follows

1popDNArand.est

```
// Priors and rules file
// *********************

[PARAMETERS]
//#isInt? #name  #dist. #min   #max
//all N are in number of haploid individuals
1  NPOP     logunif     100  100000   output bounded
1  TEXP     logunif     100  5000     output
0  RESIZE   logunif     1e-3   1000   output
0  MUTRATE  unif        1e-7   1e-9   output

[RULES]

[COMPLEX PARAMETERS]

1  ANCSIZE  =  NPOP*RESIZE    output
0  2N       =  2*NPOP         hide
0  THETA    =  2N*MUTRATE     output
```

The estimation file is divided in three main sections

1. The [**PARAMETERS**] section lists the prior distributions of simple parameters.
   Each parameter can be an *integer* or a *float*, as specified by a first indicator variable.
   Each parameter can either be uniformly or log-uniformly distributed between a minimum and a maximum value that need to be specified.
2. The [**RULES**] section can include a set of conditions to be met among the simple parameters. Rules are typically used to specify that a parameter need to be larger than another one, which could be used for instance to specify that the divergence time between two populations must be larger than the time of a bottleneck. The use of rules will of course modify the search ranges of the parameters.
3. The [**COMPLEX PARAMETERS**] lists complex parameters that are obtained as simple operations between any 2 simple or complex parameters or between a parameter and a scalar.
   The following simple operations are possible between two parameters, or between a parameter and a scalar:

   “+”, “-”, “*”, and "/"

   Since ver. 2.5.2, the following functions are possible on a single parameter or on a scalar:

   log(), log10(), exp(), abs(), pow10()

   Since ver. 2.5.2, the following functions are possible on a single parameter or between a parameter and a scalar:

   %min%, %max%

Simple or complex parameters can then be output or hidden in output files by using the keywords "output" or "hide", respectively.

The **%min%** and **%max%** functions are a bit special here as they should be used like **operators**.

Example:

```
[PARAMETERS]
1 N1   100     100000      output
1 N2   100     100000      output

[COMPLEX PARAMETERS]
1   NMAX        = N1 %max% N2     output
```

Since version 2.5.2, one can also specify that the range of a simple parameter is bounded (during parameter estimation) by using the keyword "bounded" after keywords output or hide

Example:

```
[PARAMETERS]
0 MIGR_RATE   0     0.1      output     bounded
```

This will prevent the `MIGR_RATE` parameter to grow beyond 0.1

As seen in the example `1popDNArand.est` above, comments can be included in the *est* file as double slash, as in C++.

CAUTION

Equations of complex parameters should not include any variable that contain the name of a function, i.e. log, log10, abs, pow10, min, or max (all in small caps). To avoid this problem, **it might be better to use capital letters for parameter names**.

For instance the following complex parameter NPOP1 will be incorrectly estimated

```
0   Nlog     = log(N2)     output
1   NPOP1    = exp(Nlog)   output    //Wrong: Nlog conflicts with log function
```

But it will be correctly estimated in this case

```
0   NLOG     = log(N2)     output
1   NPOP1    = exp(NLOG)   output
```

OUTPUT OF SAMPLED PARAMETERS

The sampled parameters are then output in a file having the same name as the template *tpl* file, but with the *.params* extension. Typing the following command-line

```
fsc25 -t 1popDNArand.tpl -n 10 -e 1popDNArand.est -E10
```

will draw 10 sets of parameters according to the distributions found in file *1popDNArand.est*. It will then replace the corresponding keywords found in *1popDNArand.tpl* by the values just drawn, and write a *.par* file in the output directory as well as the randomly drawn values in the *.params* file, which should look like:

./1popDNArand/1popDNArand.params

```
NPOP     TEXP          RESIZE        MUTRATE        ANCSIZE           THETA
299       396       4.1771490     9.76393e-08        1248         5.83883e-05
202       561      28.9307465     9.35616e-09        5844         3.77989e-06
4311     1062       0.0630165     8.50824e-08         271         7.33581e-04
30824     580       0.0046161     7.65865e-08         142         0.0047214
669       188      46.5000659     7.59988e-08       31108         1.01686e-04
118      2132      63.0435917     9.61273e-08        7439         2.26861e-05
49854     722      28.3119282     2.93767e-08     1411462         0.0029291
45727    2870       2.7773714     3.18337e-08      127000         0.0029113
9719      575      15.5669955     6.23843e-08      151295         0.0012126
2021      888       0.0652933     1.12363e-08         131         4.54173e-05
```

Note that the complex parameter 2N was not output here as it was tagged has hidden in the *.est* file.

Also, the command line “fsc25 *-t 1popDNArand.tpl -n 10 -e 1popDNArand.est -E10*” will generate a total of 100 arp file (10 simulations for each of the 10 sets of randomly drawn parameters), to be found in the directory *./1popDNArand*.

7. USING PREDEFINED VALUES FOR A PARTICULAR EVOLUTIONARY MODEL

fastsimcoal2 allows you to simulate data under a given evolutionary scenario with fixed and predefined values of the model parameters. This is an alternative to the generation of random parameter values seen just above, and it uses pretty much the same syntax on the command line.

For instance,

```
fsc25 -t 1popDNArand.tpl -n 5 -f 1popDNA.def
```

will replace the keywords found in the *tpl* file by the values listed in the definition file `1popDNA.def` , and it will perform five simulation for each set of defined values. Note that the *tpl* file mentioned above was used in the previous section, and we show below the content and structure of a definition file.

DEFINITION FILE

A *def* file needs to have a first header line listing the **keywords** associated to each parameter. The keywords present in the template *tpl* file must **all** be listed in the *def* file, but there can be more keywords listed in the *def* file than those listed in the *tpl* file. These additional parameters will simply not be used.

The following lines then list the values of each parameter. For each parameter set, a *par* file will be created by replacing the keywords in the *tpl* file by the corresponding parameter values. In the example below, we list just 10 sets of parameter values.

1popDNA.def

```
NPOP     TEXP       RESIZE        MUTRATE       ANCSIZE          THETA
299       396     4.1771490    9.76393e-08        1248        5.83883e-05
202       561    28.9307465    9.35616e-09        5844        3.77989e-06
4311     1062     0.0630165    8.50824e-08         271        7.33581e-04
30824     580     0.0046161    7.65865e-08         142        0.0047214
669       188    46.5000659    7.59988e-08       31108        1.01686e-04
118      2132    63.0435917    9.61273e-08        7439        2.26861e-05
49854     722    28.3119282    2.93767e-08     1411462        0.0029291
45727    2870     2.7773714    3.18337e-08      127000        0.0029113
9719      575    15.5669955    6.23843e-08      151295        0.0012126
2021      888     0.0652933    1.12363e-08         131        4.54173e-05
```

Note that this *def* file corresponds to the *params* file we had just generated from the *est* file in the previous section.

The output files (*par* and *arp* files) will be placed in a directory with the same name as the *tpl* file but without the *.tpl* extension.

8. ESTIMATING PARAMETERS FROM THE SITE FREQUENCY SPECTRUM

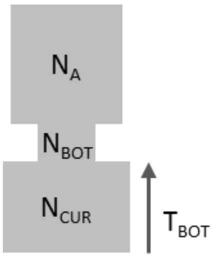
fastsimcoal2 implements a new way to estimate demographic parameters from the site frequency spectrum (SFS) computed from DNA sequence data or from ascertained SNP chips (see methodological section “Estimation of demographic parameters from the SFS”. In short, *fastsimcoal2* simulates the expected SFS under a given set of parameters and computes their (composite) likelihood. It uses a robust maximization procedure to find those parameters maximizing the composite likelihood.

In order to do this, you need the following three input files:

1. A file or a series of files containing the **observed (joint) SFS**.
2. **A template file** (***.tpl**, see Section on “Sampling parameter values from prior distributions”) specifying the evolutionary model to be studied, with a similar structure as conventional *fastsimcoal2* parameter (par) files. In this file, the parameters to be estimated are simply replaced by keywords.
3. **An estimation file** (***.est**, see chapter “Sampling parameter values from prior distributions”), which specifies the list of the search ranges for those parameters defined by their keywords corresponding to those listed in the tpl file.

EXAMPLE OF THE ESTIMATION OF A BOTTLENECK DEMOGRAPHIC HISTORY

Let's have a look at a simple example, where we want to estimate the parameters of a model where a population went through a recent bottleneck as shown in the figure below:



Note that more complex examples are provided in the Appendix section "Example files for the estimation of demography from the (joint) SFS".

OBSERVED SFS

Assume that we have the following observed SFS collected from 20Mb of DNA

1PopBot20Mb_DAFpop0.obs

```
1 observations
d0_0      d0_1  d0_2  d0_3  d0_4  d0_5  d0_6  d0_7  d0_8  d0_9  d0_10 d0_11 d0_12 d0_13 d0_14 d0_15 d0_16 d0_17 d0_18 d0_19 d0_20
19960052  9331  3572  2530  2221  2059  1963  1952  1730  1682  1572  1520  1426  1453  1335  1179  1195  1069  1129  1030  0
```

TEMPLATE FILE

Let's now define the following template file that describes the demographic model and the parameters of interests, which is similar to a par file, but where parameters are replaced by keywords.

1PopBot20Mb.tpl

```
//Number of population samples (demes)
1
//Population effective sizes (number of genes)
NCUR
//Sample sizes
20
//Growth rates : negative growth implies population expansion
```

```
0
//Number of migration matrices : 0 implies no migration between demes
0
//historical event: time, source, sink, migrants, new size, new growth rate, migr. matrix
2 historical event
TBOT 0 0 0 RESBOT 0 0
TENDBOT 0 0 0 RESENDBOT 0 0
//Number of independent loci [chromosome]
1 0
//Per chromosome: Number of linkage blocks
1
//per Block: data type, num loci, rec. rate and mut rate + optional parameters

FREQ 1 0 2.5e-8
```

Here note the FREQ data type, telling *fastsimcoal2* that we want to estimate site **frequency** spectra with simulations.

ESTIMATION FILE

The search range of the different parameters to estimate are defined in the following est file:

1PopBot20Mb.est

```
 // Priors and rules file
 // *********************
 [PARAMETERS]
 //#isInt? #name #dist.#min #max
 //all Ns are in number of haploid individuals
 1 NCUR unif 10 100000 output
 1 NANC unif 10 100000 output
 1 NBOT unif 10 100000 output
 1 TBOT unif 10 10000 output
 [RULES]
 [COMPLEX PARAMETERS]
 0 RESBOT = NBOT/NCUR hide
 0 RESENDBOT = NANC/NBOT hide
 1 TENDBOT = TBOT+100 hide
```

IMPORTANT NOTES

Even though the syntax of these est files is the same than that used when Sampling parameter values from prior distributions (see section on " Estimation file" above), some important differences exist:

1. The parameter range and distributions are not priors but rather search ranges
2. The lower range limit is an absolute minimum, whereas the upper range is only used as a maximum for choosing a random initial value for this parameter. There is actually no upper limit to the search range, as this limit can grow by 30% after each cycle if we have reached the predefined upper bound.
3. The rules in the [RULES] section are now checked during the estimation of the parameters, contrary to what happened in fastsimcoal21.

COMMAND LINE

Then parameter estimation can be simply initiated by launching the following command line

```
./ fsc25 -t 1PopBot20Mb.tpl -n 100000 -N 100000 -d -e 1PopBot20Mb.est -M 0.001 -l 10 -L 40 -q
```

This command line tells *fastsimcoal2* that the model is defined in the file *1PopBot20Mb.tpl* (-t), that the search range of the parameters to be estimated are in the file *1PopBot20Mb.est* (-e), that a minimum (-n) and a maximum (-N) of 100,000 simulations need to be done to estimate the expected derived (-d) SFS, that a minimum (-l) and maximum (-L) of 10 and 40 ECM cycles, respectively, will be performed for estimating the parameters, that the criterion (-M) to stop ECM cycles is 0.001, and that a minimum console output (quiet mode -q) is required.

Since ver 2.5.2.8, it is possible to write the command line in a file called "**fsc_run.txt**". If this file is present in the current working directory, and if *fastsimcoal2* is launched without any argument, the command line found in the file "fsc_run.txt" will be executed.

STRUCTURE OF THE FILE "FSC_RUN.TXT"

The file "fsc_run.txt" should have just two lines.

The first line should have the path to the input files. The second line should have all command line options

e.g.:

fsc_run.txt

```
D:\Users\Laurent\Dropbox\fastsimcoal\IM20Mb
-t IM20Mb.tpl -e IM20Mb.est -M0.001 -n100000 -N100000 -l10 -L20 -c6 -d -q
```

OUTPUT FILES

The results are then output in a "**<template_generic_name>.bestlhoods**" file, containing the estimated ML parameter values:

1PopExpInst20Mb.bestlhoods

```
NPOP   NANC    TEXP    MaxEstLhood    MaxObsLhood
502642  542    5070    -90349.486     -90348.375
```

In addition with estimates ML parameter values, the file reports the estimated log likelihood, as well as the maximum log likelihood given the observed SFS (MaxObsLhood). MaxObsLhood is obtained by using the observed SFS as the expected SFS when computing the likelihood.

fastsimcoal2 produces additional output files:

- "**<template_generic_name>.brent_lhoods**" containing the parameter values at different stages of the optimization procedure, after each parameter update of the ECM cycles.
- "**<template_generic_name>.last_lhoods**" containing the parameter values obtained at the last update of all ECM cycles (which might not be the ML parameter values).
- "**<template_generic_name>_maxL.par**" which is a par files where the estimated parameters have been replaced by their ML values. This file can be directly used to generate data sets corresponding to the estimated parameter values, for instance for parametric bootstrap confidence interval estimation. In that case you need to replace the FREQ keyword by DNA, to generate new SFS from the max lhood parameters.
- A file or multiple files containing the expected (joint) SFS for the ML parameters. These files have the same name as the *.obs files containing the observed SFS, but they have the *.txt extension.

OBSERVED SFS FILE NAMES

Note that the **name of the observed SFS file was not specified on the command line**. This is because it is assumed to have the **same name as the prefix of the template file** (here

1PopBot20Mb) and a given suffix, which exact definition depends on the number of population samples and on the type of SFS.

Note also that all the observed SFS files **should only contain a single observed SFS**.

ONE OBSERVED SAMPLE

If there is a single observed sample in the model, the suffix will be:

-_**DAFpop0.obs** if it is a file listing the derived allele SFS (unfolded spectrum)

-_**MAFpop0.obs** if it is a file listing the minor allele SFS (folded spectrum)

TWO OBSERVED SAMPLES

If there are two observed samples in the model (0 and 1), one would need a file with the following suffix

-_**jointDAFpop1_0.obs** if it is a file listing the derived allele SFS (unfolded spectrum)

-_**jointMAFpop1_0.obs** if it is a file listing the minor allele SFS (folded spectrum)

MORE THAN TWO OBSERVED SAMPLES

If there are more than two observed samples in the model (say 0, 1, and 2), one would need three separate files with the following suffix

-_**jointDAFpop1_0.obs, _jointDAFpop2_1.obs, _jointDAFpop2_0.obs**

For the folded spectrum, the name would begin by **_jointMAF**

MULTIDIMENSIONAL SFS

It is also possible to tell *fastsimcoal2* to use another format for observed SFS using the command line `-multiSFS`. In that case, *fastsimcoal2* expects the observed SFS to be in a single file, even when more than one population sample is specified, with the following suffix:

- **_DSFS.obs**

See the Appendix section "Multidimensional site frequency spectrum" for the exact format of this type of files.

ASCERTAINED SFS FILES

Obs files containing ascertained SFS have a suffix "-ascx.obs" instead of simply ".obs" a, where x is the ascertainment panel size (i.e. x=2 for the Affymetrix Human Origins chip panels).

9. APPENDIX

COMMAND-LINE OPTIONS

As we have already seen, *fastsimcoal2* include several command-line flags and options allowing you to parameterize simulations. These options can be listed by simply typing

`fsc25` or `fsc25 -h`

We detail the command line options in more detail below. You can use a one-letter (case sensitive) shortcut after a single dash or a longer multi-letter full option after two dashes. The value required after "some options can be separated or not from the option by a white space. For instance, "`-n1`" and "`-n 1`" are both valid.

Short cut (-)	Full (--)	Description
`-h`	`--help`	Prints a list of all available options
`-i`	`--ifile`	Name of parameter file. Example: `-i 1popDNA.par`
`-n`	`--numsims`	Number of simulations to perform per parameter file or sets of parameter (see -e option) Example: `-n 1000`
`-t`	`--tplfile test.tpl`	Name of template parameter file. The template file must have the extension *.tpl* Example: `-t 1popDNArand.tpl`
`-f`	`--dfile test.def`	Name of a parameter definition file. This file includes a list of sets of parameter values to be substituted in the *tpl* file. Listed parameter values are substituted in the template file to produce a valid parameter file (.par). This is an alternative to the random drawing of parameter values from distributions defined in the *est* file. Example: `-f 1popDNArand.def`
`-F`	`--dFile test.def`	Same as *-f* option, but only it only uses the simple parameters defined in *est* file. Complex parameters are recomputed from the simple parameters. Note that this option must be used together with both the *-t* and the *-e* options. The *est* file is here necessary to define the relationships between the simple parameters. Example: `-F 1popDNArand.def`
`-e`	`--efile test.est`	Name of parameter prior definition file. Parameters are drawn from specified distributions and then substituted into template file. The file must have the extension *.est* . A template file must be provided for this option to work. Example: `-e 1popDNArand.est`

`-E   --numest`

Number of sets of parameters to draw from the prior distributions defined in the *est* file. Parameter can be omitted for FREQ data type.

Example: `-e 1popDNArand.est -E 100`

`-g   --genotypic`

Generates *Arlequin* projects outputs (*.*arp*) in genotypic (diploid) format. Without this option, *arp* file are in the haploid format.

`-p   --phased`

Specifies that phase is known in *Arlequin arp* output files. Without this options phase is assumed unknown.

`-s   --dnatosnp 2000`

Output DNA as SNP data, with a given maximum number to output (use 0 to output all SNPs in the DNA sequence(s)).

`-S   --allsites`

Output the whole DNA sequence, including all monomorphic sites.

`-I   --inf`

Generates DNA mutations according to an infinite site (IS) mutation model. Under this model, each mutation is supposed to occur at a different but random site on the DNA sequence. Under the IS model, if different mutations are allocated to the same DNA sequence position, they are generated independently, but marked to have occurred at the same position in the *Arlequin* output *arp* file.

`-d   --dsfs`

Computes the site frequency spectrum (SFS) for the **derived** alleles for each population sample and for all pairs of samples (joint 2D SFS). Note that this is only done on SNP or DNA data. If your input file is about DNA sequences, then use the *-s* option to convert DNA to SNP data.

`-m   --msfs`

Computes the site frequency spectrum (SFS) for the **minor** allele for each population sample, for all pairs of samples (joint SFS), and for all populations samples pooled (global SFS). Note that this is only done on SNP data. If your input file is about DNA sequences, then use the *-s* option to convert DNA to SNP data.

`-j   --jobs`

output one simulated or bootstrapped SFS per file
in a separate directory for easier analysis
(requires -d or -m and -s0 options)

`-b   --nonparboot 10`

number of bootstraps to perform on polymorphic sites to extract SFS (should be used in addition to -s0 and -j options)

`-H   --header`

Generates a header in the site frequency spectrum (SFS) files

`-q   --quiet`

Outputs minimal messages to the console instead of detailed messages

`-T   --tree`

Output coalescent tree for each non-recombining segment in nexus format. With recombination, a tree is output for each of the segment between recombination breakpoints.

`-k   --keep 100000`

Number of simulated polymorphic sites to keep in memory before writing them to temporary files. If this option is not specified, a default value of 200000 used. This option is mostly useful when generating relatively long DNA sequences, with potentially tens of thousands of polymorphic sites. In that case, extended k to a large value will allow you to keep all genetic information in memory and slightly speed up computations. Note that setting up this *k* value to a very large number may imply prohibitive memory requirement for *fastsimcoal2,* especially if you have large sample sizes i.e. thousands of individuals). In that case, it is preferable to adjust k, to prevent *fastsimcoal2* to use all your computer memory. In

		general the default value of *k*= 200000 gives very good results and does not need to be changed. Note that larger values could be used when estimating the DSFS or MSFS from large DNA sequence data. Example: `-k 10000`
`-r`	`--seed`	Seed for random number generator (positive integer <= 1E6)
`-x`	`--noarloutput`	Does not generate Arlequin output files
`-D`	`--dadioutput`	Output SFS computed from DNA sequences in dadi format.
`-M`	`--maxlhood 0.01`	Perform parameter estimation by maximum composite likelihood from the SFS, and defines stopping criterion as minimum relative difference in parameters between two iterations Example: `-M 0.01`
`-N`	`--maxnumsims 100000`	Maximum number of simulations to be performed when estimating parameters from SFS. The minimum number is defined by the `-n` option, and 5000 more simulations are performed after each simulation. Default is 100,000. Example: `-n 10000 -N 50000`
`-l`	`--minnumloops 10`	Minimum number of loops (ECM cycles) to be performed when estimating parameters from SFS. Default is 10. Example: `-l 5`
`-L`	`--maxnumloops 20`	Maximum number of loops (ECM cycles) to be performed when estimating parameters from SFS. Default is 20. Example: `-L 30`
`-C`	`--minSFSCount 1`	Minimum observed SFS entry count taken into account for parameter estimation (default = 1)") Example: `-C 5`
`-0`	`--removeZeroSFS`	Does not take into account monomorphic sites in observed SFS for parameter inference. Parameters are estimated only from the SFS and mutation rate is ignored. This option requires that one parameter be fixed in the est file.
`-a`	`--ascDeme 0`	Deme id where ascertainment is performed when simulating ascertained SNPs. Do not use this option for simulating unascertained SNP. Note that this ascertainment differs from that based on minimum frequency when generating SNP. This ascertainment applies specifically to DNA sequences. Example: `-a 0`
`-A`	`--ascSize 2`	Number of ascertained chromosomes used to validate SNPs. Minimum value is 2. Default = 2. Example: `-A 2`
`-u`	`--multiSFS`	Generates or use multidimensional SFS. It implies a different format than when 1D or 2D SFS are used, even though this option can also be used when modeling only 1 or 2 populations.
`-w`	`--brentol 0.01`	Tolerance level for Brent optimization Default = 0.01. Smaller value imply more precise estimations, but require more computation time (min;max) = (1e-1;1e-5)
`-c`	`--cores 1`	Number of openMP threads to be used for simulation of independent chromosomes and for parameter estimation

(default=1, max=numBatches, use 0 to let openMP choose optimal value). Note that this option is only active for 64 bit version of the program.

Example: `-c0`

`-B --numBatches 12` Maximum number of batches for multi-threaded runs (default=12). For multithreaded runs, the simulations are divided into a specified number of batches, each executed in a given thread. Best performance is achieved when the number of batches is equal to the number of cores available. Note that simulations and estimation will be the same if different numbers of threads are used with the same number of batches. Different results will be obtained with the same seed for different no. of batches, even if the same number of cores is used.

Example: `-c4 -B4`

`-P --pooledsfs` computes pooled SFS over all samples

Assumes `-d` or `-m` , but not `-u` (--multiSFS) flag activated

`--recordMRCA` records tMRCAs for each non recombining segment and outputs results in file <generic name>_mrca.txt. Beware: huge slow down of computing time

MULTITHREADING

New to fastsimcoal ver 2.5, multithreading is introduced via the use of openMP threads. This has required considerable changes in most C++ classes to make them thread safe. It has also required the introduction of a new thread-safe way to generate random numbers, so that simulations or parameter estimations would give identical results independently of the number of threads used.

Multithreading has been implemented at two levels: when simulating multiple chromosome segments and when estimating model parameters from the SFS. It is thus not implemented when generating a single large recombining segment, as there is no way to split this simulation into independent units. Due to openMP limitations, it is also not implemented in the windows 32bit version.

In summary, a given workload is divided into a predefined number of batches that are processed by independent threads, and results are then collected once all threads are finished. The default number of batches is 12, and the default number of threads is 1. The number of threads can be specified by the command line option -c, and the number of batches by the option -B.

In the figures below, we report the computing time of two tasks as a function of the number of batches and of the number of threads used. Note that the computations were done on a quad core system with hyperthreading activated, resulting in 8 potential independent threads visible to win8.1.

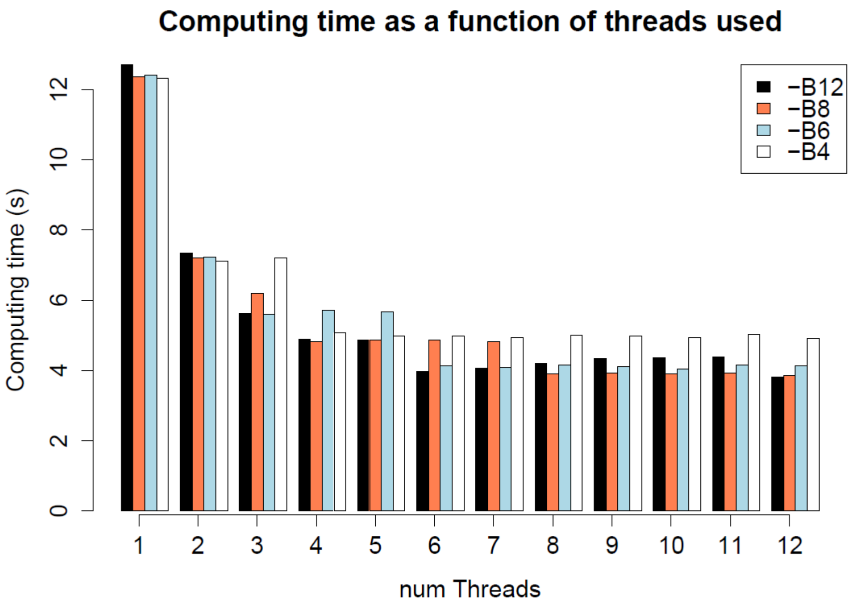
The biggest observed computing gain is to pass from 1 to 2 threads, and then there is a clear diminishing return in speed gain. The figures also show that the highest speed is obtained when the number of batches is equal to the number of threads. In that case the highest performance should be obtained when the number of threads is equal to the number of available cores (8 in our case), but in practice we have a slightly better performance when using 6 threads and 12 batches (as the same threads are each used twice). Actually, the best performance we obtained overall was when using 8 cores and 16 batches (not shown). Note that you can use the option -c0 to let openMP decide on the maximum number of cores to use if you do not know this for your machine. What is also clear is that performance does not change much once the number of threads exceeds the number of batches. In summary, the use of 6-8 threads on most modern desktops should give a good performance increase.

When performing computations on a linux cluster, and admitting that one needs to replicate runs from different starting points (min=50) to get the global maximum likelihood, then best performance should be obtained by using a single thread and using one batch (-B1). As the speed gain should be exactly equal to the number of available cores.

The bash file (*fsc_threads_speed_test.sh*) used to compare computing times is available in the "example files" directory, and can be used on other systems to find best combinations of -c and -B options

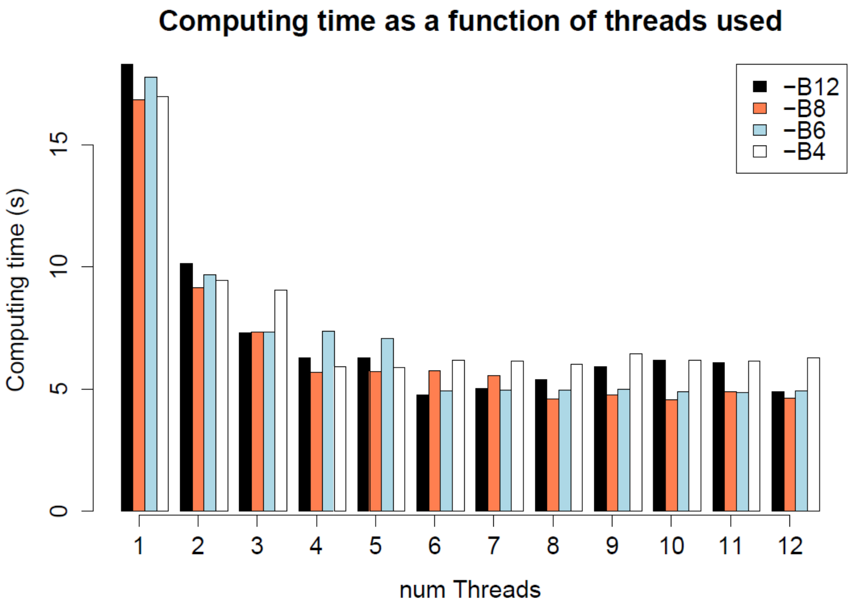
./$fsc25 -i 1PopDNAnoRec10Mb.par -n10 -I -x --seed 1234 - 10 simulations of 100,000 segments of 100 bp

Additional options: -c1 to -c12 and -B4 to -B12



./$fsc -t 1PopExpInst20Mb.tpl -e 1PopExpInst20Mb.est -d -M0.001 -n200000 -N200000 -I -l5 -L5 --seed 1234 -q

Additional options: -c1 to -c12 and -B4 to -B12



SEQUENTIAL MARKOV COALESCENT APPROXIMATION

In this section, we briefly describe the principle of the sequential Markov Coalescent model that is used to approximate the classical ancestral recombination graph (ARG). For more details, please read the original publications (McVean and Cardin 2005, Marjoram and Wall 2006, Chen et al. 2009).

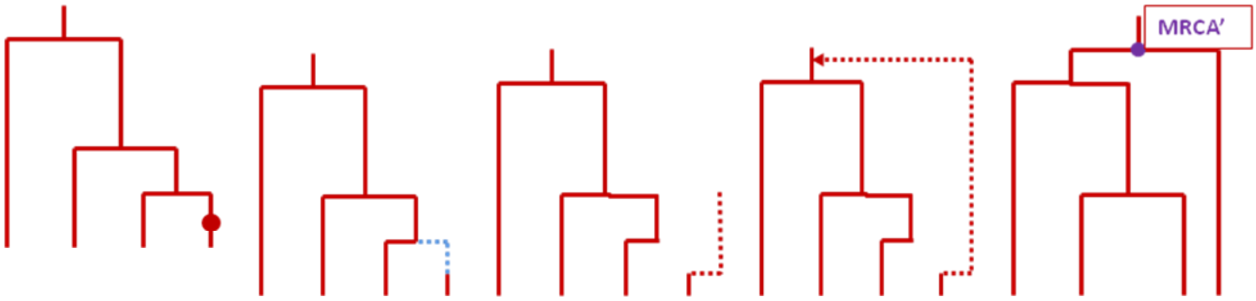
In brief, McVean and Cardin (2005) proposed to approximate the coalescent with recombination over a chromosomal segment based on the ARG by simulating a series of trees that differ each by single recombination events, starting on the left of the segment and moving to the right of the segment.

They thus proposed the following SMC algorithm to simulate coalescent with recombination along a DNA segment of length L :

1) Simulate a normal coalescent tree without recombination on the left hand side of the segment, and set $x = 0$. This initial tree has a total size of $T = T_0$ generations.
2) Select the position of the next recombination event on the segment by drawing a random number $y \sim \mathrm{Exp}(rT)$, where Exp is an exponential distribution, and *r* is the recombination rate per generation between adjacent base pairs.
3) If $x + y < L$, select a position on the current tree where to implement a recombination event, by drawing a uniform number in $1..T$.
4) Detach the branch below the recombining event from the tree, and color in blue the branch between the recombination event and the next interior node.
5) Erase the blue branch.
6) Let the recombining lineage coalesce with the other lineages attached to the tree, and thus reattach the detached tree to the left tree.
7) Record the total length of the new tree T' and set $T = T'$.
8) Let $x+ = y$
9) Repeat steps 2)-8) until $x + y \geq L$.

The steps 3 to 6 of the SMC algorithm are illustrated below

Illustration of the SMC algorithm



A recombination event ● occurs on a tree

The (blue) branch above the recombination event is erased

The recombining lineage is free to evolve backward in time …

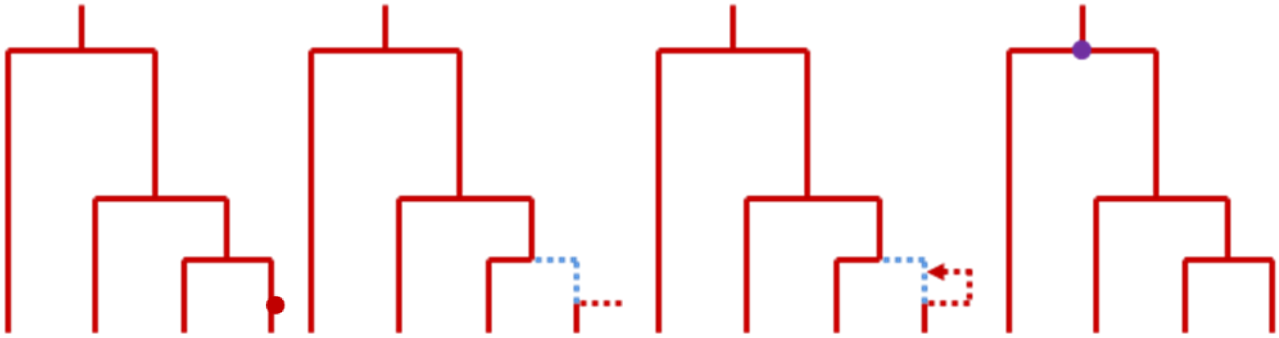
… until it coalesces with one of the lineage on the left tree …

… potentially changing the tree topology and the MRCA of the right tree

Marjoram and Wall (2006) proposed a slight modification of the SMC algorithm consisting in removing step 5 of the SMC algorithm above, and thus keeping the blue branch, and thus leaving the possibility to the recombining lineage to coalesce with it. The blue branches are only erased at the end of the simulation. This SMC' algorithm has been shown to give patterns of linkage disequilibrium more similar to those obtained by *ms* (Marjoram and Wall 2006), and it is therefore this algorithm that is implemented in *fastsimcoal2* .

With SMC', the recombining lineage can indeed coalesce with the blue lineages, and thus lead to a tree identical to the left tree, reflecting the fact that in the ancestral recombination graph, coalescent event can also occur between unsampled lineages.

Illustration of the SMC' algorithm



A recombination event ● occurs on a tree

The blue branch above the recombination event is kept

The recombining lineage is free to evolve backward in time, and can coalesce with the blue branch …

… leading to a tree identical to that on the left.

SITE FREQUENCY SPECTRUM

fastsimcoal2 can generate the site frequency spectrum for the derived allele (the unfolded site frequency spectrum) of the simulated data sets with the *-s* and *-d* command-line options as follows in the case of the relatively complex 3-population scenario described above:

```
fsc25 -i 3PopDNASFS.par -x -s0 -d -n10 -q
```

The *-s* option is necessary, here because the SFS can only be computed on SNP data, and the *-s* options tells *fastsimcoal2* to treat DNA as SNP data (where the 0 allele is ancestral). Note that the *-x* option is there to prevent the output of *arlequin* input files, which would be quite large in this case (about 2.8 Mb for each simulation, with >50,000 polymorphic sites).

fastsimcoal2 will then generate for each simulation, the population-specific derived site frequency spectrum, all the pairwise joint frequency spectra.

For instance, the 10 SFS of deme 0 look like:

./3PopDNASFS/3PopDNASFS_DAFpop0.obs

```
10 observations
d0_0      d0_1  d0_2  d0_3  d0_4  d0_5  d0_6  d0_7  d0_8  d0_9 d0_10 d0_11 d0_12 d0_13 d0_14 d0_15 d0_16 d0_17 d0_18 d0_19 d0_20
9964672   8054  4046  3032  2190  2118  1548  1431  1358  1138  1096  1010  1020   845   688   803   673   743   668   614  2253
9964237   8407  4203  2833  2675  1904  1587  1443  1241  1093  1073   923   933   917   855   752   837   590   637   618  2242
9963855   8449  4104  3165  2700  2045  1662  1634  1267  1067   904   897   923   930   844   845   808   696   490   556  2159
9964355   8292  4251  3142  2248  1741  1552  1325  1359  1290  1147  1068   938   851   846   768   632   747   626   728  2094
9963918   8582  4285  3110  2391  2081  1506  1475  1279  1119   967   986   929   841   763   797   841   752   651   687  2040
9964418   8409  4483  2844  2241  1896  1580  1407  1307  1109   978  1015   857   821   720   777   697   599   654   647  2541
9965301   7894  4487  3061  2212  1719  1562  1384  1154  1059   962   992   800   799   785   673   715   664   700   571  2506
9963601   8491  4372  3120  2547  2090  1525  1450  1336  1164  1018   877   945   844   741   849   808   768   705   591  2158
9965241   7855  4273  2918  2294  1960  1503  1362  1103  1023   996   965   793   870   810   863   632   804   571   679  2485
9964845   8306  4674  2928  2205  1828  1519  1414  1142  1203  1062  1010   900   882   785   656   654   581   581   552  2273
```

Whereas this is the derived SFS, it has to be noted that the derived allele can have here a frequency of 20/20 since we use here a finite site model allowing more than one mutation per site. So by chance two mutations might have occurred in the two descending branches stemming from the

MRCA node and lead to the same derived allele, implying that the derived allele frequency will be 20. To prevent this, use the `-I` option that generates SNPs under an infinite site model. Note that we also output here the SFS entry 0/0, which just lists the number of monomorphic sites in deme 0.

In the following table we show the first two simulated joint SFS between deme 0 and deme 2

./3PopDNASFS/3PopDNASFS_DAFpop0.obs

```
10 observations
        d0_0   d0_1 d0_2 d0_3 d0_4 d0_5 d0_6 d0_7 d0_8 d0_9 d0_10 d0_11 d0_12 d0_13 d0_14 d0_15 d0_16 d0_17 d0_18 d0_19 d0_20
d2_0 9952704   7659 3669 2686 1832 1709 1214 1048  960  811   729   689   689   537   389   453   403   386   317   285  1329
d2_1    3558     32   27   25   22   38   19   25   36   21    19    22    21    35    22    23    24    27    22    24   141
d2_2    2359     38   38   29   46   14   28   36   36   26    31    23    33    23    32    23     8    10    28    32   177
d2_3    1341     37   58   29   31   63   27   38   37   28    15    17    31    27    16    38    23    21    43    23   139
d2_4    1059     38   28    8   24   27   43   34   36   38    43    21    25    26    21    13    46    24    17    22   159
d2_5     830     26   15   29   19   33   34   33   28   16    12    19    16    17    12    16    16    15    13    25   171
d2_6    2821    224  211  226  216  234  183  217  225  198   247   219   205   180   196   237   153   260   228   203   137
d2_0 9951575   8022 3824 2506 2278 1569 1249 1100  911  788   696   572   573   525   483   389   404   316   353   287  1160
d2_1    4443     50   28   38   28   44   38   26   25   34    57    40    42    42    32    35    30    21    28    24   183
d2_2    2278     33   45   42   59   30   25   41   29   25    26    17    42    36    34    26    29    26    24    27   205
d2_3    1422     56   19   14   20   19   16   26   26   36    12    25    27    26    37    24    16     8    16    27   170
d2_4    1185     30   29   36   34   44   39   38   35   27    43    35    19    28    27    52    55    24    33    15   220
d2_5     926     30   44   37   41   33   18   39   42   34    65    23    44    35    26    25    21    31    23    25   217
d2_6    2408    186  214  160  215  165  202  173  173  149   174   211   186   225   216   201   282   164   160   213    87
```

The other SFS are then just listed below in the file `./3PopDNASFS/3PopDNASFS_DAFpop0.obs` Again, the SFS entries (n_2,j) and (i,n_0) are also listed here as some derived alleles can be fixed in either or both populations if the defining mutation has occurred on some lineages leading exclusively to genes sampled in the third population. Finally, note that SFS cannot be computed on DNA sequences presenting more than a few 10s of thousand polymorphic sites, due to prohibitive memory requirements, and the need to write temporary files with intermediate results on the computer. But for the 3-population model defined above, this would imply the simulation of sequences larger than about 200Mb. However, you can increase the number of polymorphic sites to keep in memory with the -k command line option. If your computer has enough memory, then you can probably generate and compute the SFS on a very large data set with >1 million SNPs.

It can be important to realize that the computation of SFS can perfectly be done on simulated data obtained for random values of the parameters, with the use of *tpl* and *est* files, thus allowing one to get the empirical distribution of SFS under a given model. These SFS could then be perfectly used as summary statistics to estimate parameters from an observed SFS under an ABC framework or for model choice (see below), but they can also be used to infer parameters via likelihood maximization (as discussed below).

MINOR ALLELE SITE FREQUENCY SPECTRUM

Minor allele SFS for multiple populations have now a modified format. Whereas the Derived allele SFS is straightforward to compute, the Minor allele SFS is not when several populations are involved. The observed (and the expected) MAF SFS are now computed given the following rules.

a) The minor allele is identified after computing the global frequency of the two alleles over all samples.

b) For each site where the minor allele frequency (MAF) is equal to the major allele frequency, we cannot decide which allele is the minor allele, and thus, the observed SFS entry of each of the two possible minor allele should be updated by 0.5.

MULTIDIMENSIONAL SITE FREQUENCY SPECTRUM

With the option `--multiSFS`, you have the option to generate multidimensional site frequency spectrum.

For example if you use the following par file

./3PopDNASFSsmall2.par

```
//Number of population samples (demes)
3
//Population effective sizes (number of genes)
20000
5000
10000
//Sample sizes
2
3
6 1500
//Growth rates: negative growth implies population expansion
0
0
0
//Number of migration matrices : 0 implies no migration between demes
0
//historical event: time, source, sink, migrants, new size, new growth rate, migr. matrix
4  historical event
2000 1 2 0.05 1 0 0
2800 1 1 0 0.04 0 0
3000 1 0 1 1 0 0
15000 0 2 1 3 0 0
//Number of independent loci [chromosome]
1 0
//Per chromosome: Number of linkage blocks
1
//per Block: data type, num loci, rec. rate and mut rate + optional parameters
DNA 10000000 0.00000001 0.00000002 0.33
```

to produce multidimensional SFS with the following command line

```
fsc25 -i 3PopDNASFSsmall.par -x -s0 -d -n1 --multiSFS --seed 1234 -q
```

you will generate the following observed SFS

./3PopDNASFSsmall/3PopDNASFSsmall2_DSFS.obs

```
1 observations. No. of demes and sample sizes are on next line
3                2    3    6
9964343  4506    2046 1320  1154  873   3229  1872   50      30      15      29
41       190     982  22    17    20    20    36     179     1936    86      66
63       43      28   371   5506  68    54    73     91      110     618     350
16       22      15   31    7     92    415   13     26      7       19      16
74       1666    86   85    87    42    60    604    1184    41      48      53
55       94      294  286   23    3     4     7      17      58      298     19
12       2       14   14    97    2494  190   188    203     225     255     2
```

Note that the second line gives the number of populations and the sample sizes in each deme. Note also that the whole **3D SFS is normally produced on a single line** (line 3), but it was split here on 7 lines for better visualization.

The entries on the SFS above correspond to the number of sites having the following derived allele frequencies

3rd line: (0,0,0), (0,0,1), (0,0,2), (0,0,3), (0,0,4), (0,0,5), (0,0,6), (0,1,0), (0,1,1), (0,1,2), (0,1,3), (0,1,4)

4th line: (0,1,5), (0,1,6), (1,2,0), (1,2,1), (1,2,2), (1,2,3), (1,2,4), (1,2,5), (1,2,6), (1,3,0), (1,3,1), (1,3,2)

5th line: (1,3,3), (1,3,4), (1,3,5), (1,3,6), (2,0,0), (2,0,1), (2,0,2), (2,0,3), (2,0,4), (2,0,5), (2,0,6), (2,1,0)

…

9th line: (2,2,2), (2,2,3), (2,2,4), (2,2,5), (2,2,6), (2,3,0), (2,3,1), (2,3,2), (2,3,3), (2,3,4), (2,3,5), (2,3,6)

Note as well that the entries of the matrix are in the form (x_0, x_1, x_2), where x_0, x_1, and x_2 are the derived allele frequencies in demes 0, 1, and 2, respectively. We thus start iterating allele frequencies in demes 2, then in deme 1 and finally in deme 0, which might be counter-intuitive, but this format has been used in dadi before, and we kept it for maintaining compatibility.

This single file captures all the possible allelic configurations in these three samples, which are only marginalized in the 3 joint SFS that would be produced by the command

```
fsc25 -i 3PopDNASFSsmall.par -x -s0 -d -n1 -seed 1234 -q
```

which generates the three joint SFS files

./3PopDNASFSsmall/

3PopDNASFSsmall_jointDAFpop1_0.obs

```
1 observations
        d0_0     d0_1 d0_2
d1_0    9976936  6160 2036
d1_1    2866     644  519
d1_2    1435     650  618
d1_3    2409     2469 3258
```

3PopDNASFSsmall_jointDAFpop2_0.obs

```
1 observations
        d0_0     d0_1 d0_2
d2_0    9969356  7535 4048
d2_1    4210     218  299
d2_2    2435     157  307
d2_3    1735     247  325
d2_4    1162     149  354
d2_5    1106     226  465
d2_6    3642     1391 633
```

3PopDNASFSsmall_jointDAFpop2_1.obs

```
1 observations
        d1_0     d1_1 d1_2 d1_3
d2_0    9970975  2918 1926 5120
d2_1    4196     85   55   391
d2_2    2369     124  76   330
d2_3    1742     124  64   377
d2_4    1155     93   89   328
d2_5    1157     103  40   497
d2_6    3538     582  453  1093
```

Either multiple joint 2DSFS can be used as input for parameter inference from SFS, and not that the resulting likelihood will differ according to the format we choose, as it will be computed in different ways (see equations 4 and 7 below). The input file format is automatically adjusted with the presence or absence of the `--multiSFS` option. Note however that when more than three populations are used most of the entries of the multidimensional SFS are empty, which may cause estimation problems due to imprecisions in the estimation the expected probabilities of these entries. See section on parameter inference "Composite likelihoods".

GENERATING SFS IN SINGLE FILES

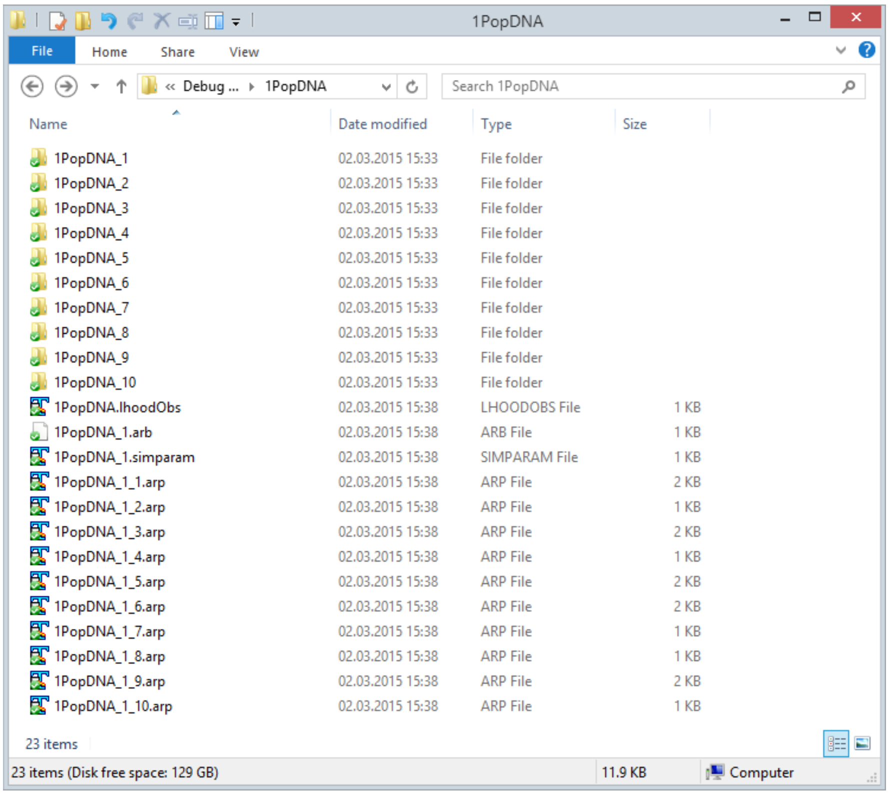
It is possible to ask fastsimcoal to put the computed SFS in separate directories, for later easier analyses, for instance for the computation of bootstrap confidence intervals after some parameter estimation. This is done with the `-j` option, which should be associated with the `-m` or `-d` and `-s0` options

For instance, the command

```
./fss25 -i 1PopDNA.par -n10 -q -j -s0 -d
```

will generate 10 arp files, and will compute their associated derived allele SFS, and each SFS file will be put in a different directory, for easier parallel analysis on a cluster later on.

The result 1PopDNA directory will look like:

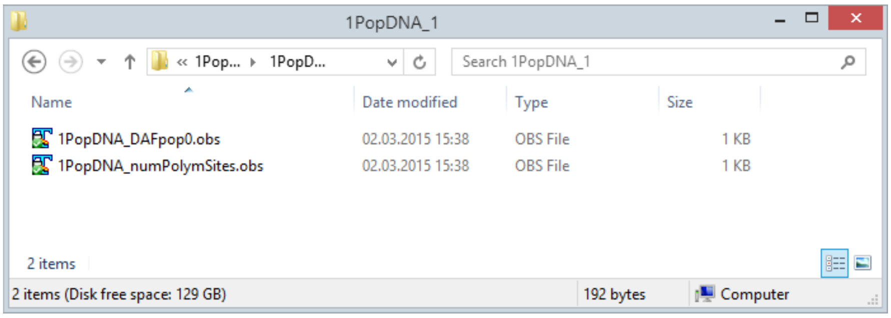


Name	Date modified	Type	Size
1PopDNA_1	02.03.2015 15:33	File folder	
1PopDNA_2	02.03.2015 15:33	File folder	
1PopDNA_3	02.03.2015 15:33	File folder	
1PopDNA_4	02.03.2015 15:33	File folder	
1PopDNA_5	02.03.2015 15:33	File folder	
1PopDNA_6	02.03.2015 15:33	File folder	
1PopDNA_7	02.03.2015 15:33	File folder	
1PopDNA_8	02.03.2015 15:33	File folder	
1PopDNA_9	02.03.2015 15:33	File folder	
1PopDNA_10	02.03.2015 15:33	File folder	
1PopDNA.lhoodObs	02.03.2015 15:38	LHOODOBS File	1 KB
1PopDNA_1.arb	02.03.2015 15:38	ARB File	1 KB
1PopDNA_1.simparam	02.03.2015 15:38	SIMPARAM File	1 KB
1PopDNA_1_1.arp	02.03.2015 15:38	ARP File	2 KB
1PopDNA_1_2.arp	02.03.2015 15:38	ARP File	1 KB
1PopDNA_1_3.arp	02.03.2015 15:38	ARP File	2 KB
1PopDNA_1_4.arp	02.03.2015 15:38	ARP File	1 KB
1PopDNA_1_5.arp	02.03.2015 15:38	ARP File	2 KB
1PopDNA_1_6.arp	02.03.2015 15:38	ARP File	2 KB
1PopDNA_1_7.arp	02.03.2015 15:38	ARP File	1 KB
1PopDNA_1_8.arp	02.03.2015 15:38	ARP File	1 KB
1PopDNA_1_9.arp	02.03.2015 15:38	ARP File	2 KB
1PopDNA_1_10.arp	02.03.2015 15:38	ARP File	1 KB

23 items

23 items (Disk free space: 129 GB) 11.9 KB Computer

and each subdiretory will contain a separate SFS file, like:



Name	Date modified	Type	Size
1PopDNA_DAFpop0.obs	02.03.2015 15:38	OBS File	1 KB
1PopDNA_numPolymSites.obs	02.03.2015 15:38	OBS File	1 KB

2 items

2 items (Disk free space: 129 GB) 192 bytes Computer

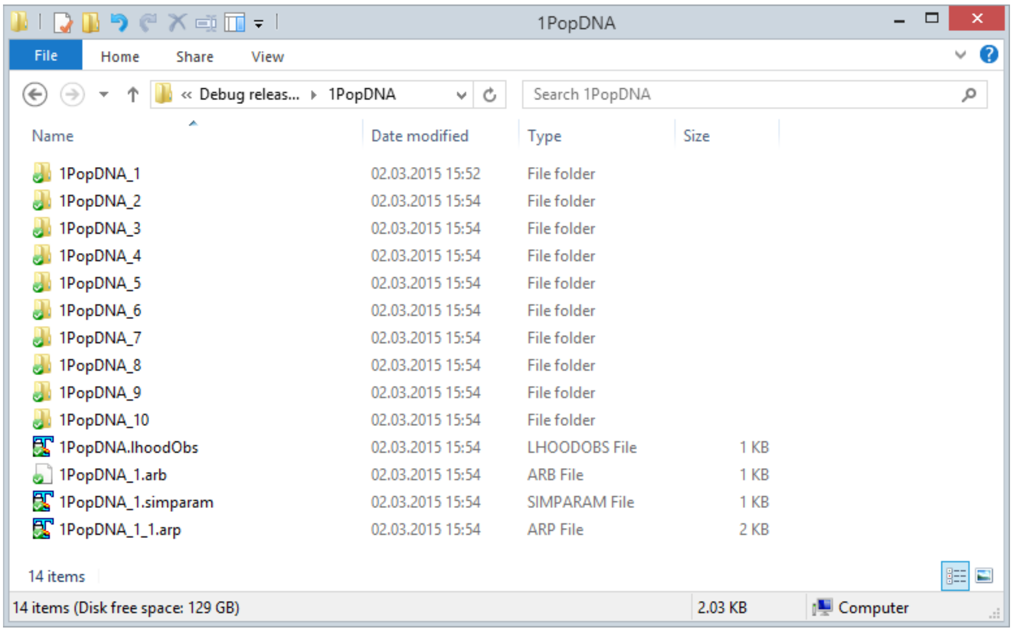
GENERATING NON-PARAMETRIC BOOTSTRAPED SFS

In addition to the generation of a given number of simulated SFS, it is possible to output, for each generated SFS, a number of bootstrapped SFS, where polymorphic sites are drawn with replacement from those obtained in the simulations.

This is obtained by using the `-b` option, like :

```
./fsc25 -i 1PopDNA.par -n1 -q -j -s0 -d -b9
```

In that case it will generate a single arp file, but 10 directories. The first directory "`1PopDNA_1`" will contain the SFS computed on the original data, whereas the 9 others will contain the SFS computed on the bootstrap data.



Name	Date modified	Type	Size
1PopDNA_1	02.03.2015 15:52	File folder	
1PopDNA_2	02.03.2015 15:54	File folder	
1PopDNA_3	02.03.2015 15:54	File folder	
1PopDNA_4	02.03.2015 15:54	File folder	
1PopDNA_5	02.03.2015 15:54	File folder	
1PopDNA_6	02.03.2015 15:54	File folder	
1PopDNA_7	02.03.2015 15:54	File folder	
1PopDNA_8	02.03.2015 15:54	File folder	
1PopDNA_9	02.03.2015 15:54	File folder	
1PopDNA_10	02.03.2015 15:54	File folder	
1PopDNA.lhoodObs	02.03.2015 15:54	LHOODOBS File	1 KB
1PopDNA_1.arb	02.03.2015 15:54	ARB File	1 KB
1PopDNA_1.simparam	02.03.2015 15:54	SIMPARAM File	1 KB
1PopDNA_1_1.arp	02.03.2015 15:54	ARP File	2 KB

14 items

14 items (Disk free space: 129 GB) 2.03 KB Computer

CAUTION AND USE OF BLOCK-BOOTSTRAP WITH REAL DATA

With real data, non-parametric bootstraps are usually obtained by resampling observed sites and then recomputing the SFS. If you have some linkage disequilibrium in your data, then it might be a good idea to implement a block-bootstrap strategy, where, instead of bootstrapping single sites, you would be bootstrapping whole chromosome segments, which would include several sites at a time. The size of the segments should be chosen such as that LD between endpoints would be almost nil.

GENERATING PARAMETRIC BOOTSTRAPS SFS

I describe below how to generate parametric bootstraps to get confidence intervals after parameter estimation.

As mentioned above, fsc creates a file with the extension **_maxL.par* in the result folder. This par file is created from the maximum likelihood parameters estimated during the optimization and estimation procedure. This file shown below can be used to perform parametric bootstrap.

1PopExpInst20Mb_maxL.par

```
//Parameters for the coalescence simulation program : fastsimcoal2.exe
1 samples to simulate :
//Population effective sizes (number of genes)
505946
//Samples sizes and samples age
10
//Growth rates: negative growth implies population expansion
0
//Number of migration matrices : 0 implies no migration between demes
0
//historical event: time, source, sink, migrants, new deme size, new growth rate, migration matrix index
1      historical event
5079 0 0 0 1.03173066e-003 0 0
//Number of independent loci [chromosome]
1 0
//Per chromosome: Number of contiguous linkage Block: a block is a set of contiguous loci
1
//per Block:data type, number of loci, per generation recombination and mutation rates and optional parameters
FREQ   1    0    2.5e-8 OUTEXP
```

This file needs to be modified to generate DNA sequence data (here 200,000 non-recombining segments of 100 bp) as

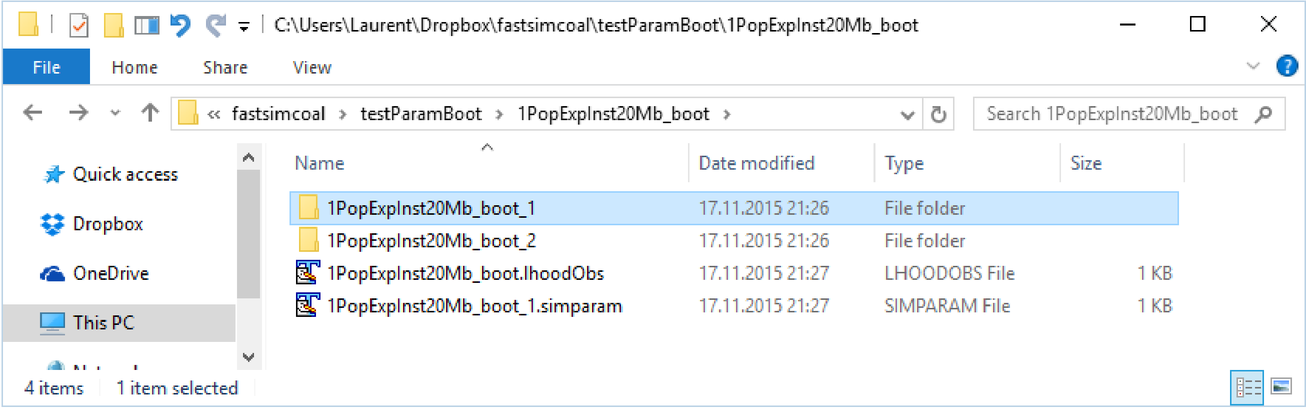
1PopExpInst20Mb_boot.par

```
//Parameters for the coalescence simulation program : fastsimcoal2.exe
1 samples to simulate :
//Population effective sizes (number of genes)
505946
//Samples sizes and samples age
10
//Growth rates: negative growth implies population expansion
0
//Number of migration matrices : 0 implies no migration between demes
0
//historical event: time, source, sink, migrants, new deme size, new growth rate,
migration matrix index
1     historical event
5079 0 0 0 1.03173066e-003 0 0
//Number of independent loci [chromosome]
200000 0
//Per chromosome: Number of contiguous linkage Block: a block is a set of contiguous
loci
1
//per Block:data type, number of loci, per generation recombination and mutation rates and optional parameters
DNA  100   0   2.5e-8 OUTEXP
```

Then running the following command

```
./fsc25 -i 1PopExpInst20Mb_boot.par -n2 -j -d -s0 -x -I -q
```

will generate 2 SFS in two separate subdirectories in the result directory *1PopExpInst20Mb_boot*



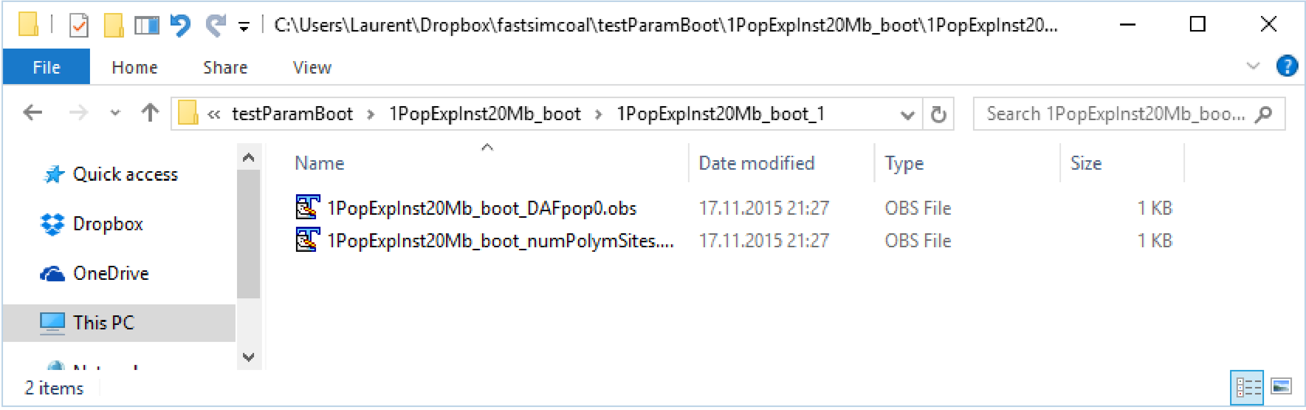
C:\Users\Laurent\Dropbox\fastsimcoal\testParamBoot\1PopExpInst20Mb_boot

« fastsimcoal › testParamBoot › 1PopExpInst20Mb_boot ›

Search 1PopExpInst20Mb_boot

Name	Date modified	Type	Size
1PopExpInst20Mb_boot_1	17.11.2015 21:26	File folder	
1PopExpInst20Mb_boot_2	17.11.2015 21:26	File folder	
1PopExpInst20Mb_boot.lhoodObs	17.11.2015 21:27	LHOODOBS File	1 KB
1PopExpInst20Mb_boot_1.simparam	17.11.2015 21:27	SIMPARAM File	1 KB

4 items   1 item selected



C:\Users\Laurent\Dropbox\fastsimcoal\testParamBoot\1PopExpInst20Mb_boot\1PopExpInst20...

« testParamBoot › 1PopExpInst20Mb_boot › 1PopExpInst20Mb_boot_1

Search 1PopExpInst20Mb_boo...

Name	Date modified	Type	Size
1PopExpInst20Mb_boot_DAFpop0.obs	17.11.2015 21:27	OBS File	1 KB
1PopExpInst20Mb_boot_numPolymSites....	17.11.2015 21:27	OBS File	1 KB

2 items

With this procedure one can easily generate more than 2 SFS for these sets of parameters (say 100) and then estimate parameters from these pseudo-observed data sets using the same *tpl* and *est* files as those used to get the **_maxL.par* file.

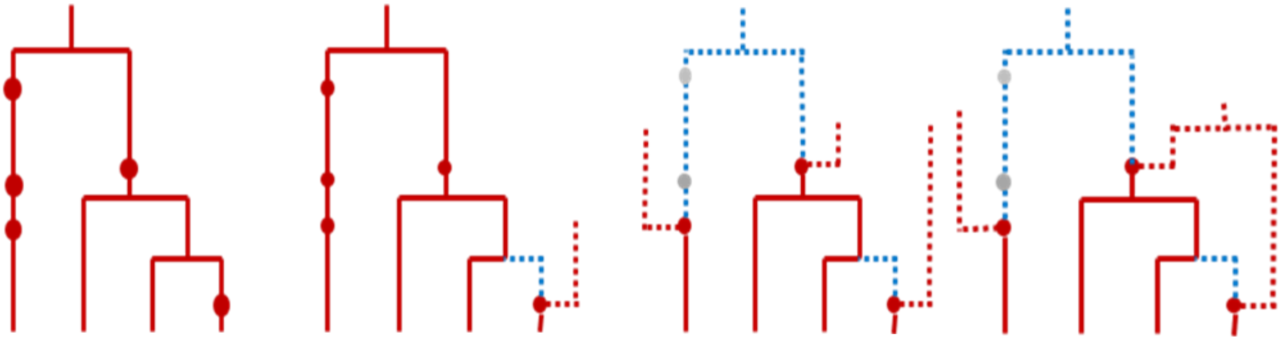
EXTENSION OF THE SMC' ALGORITHM TO MULTIPLE RECOMBINATION EVENTS

In order to simulate markers located at fixed recombination distances on the chromosome, we have extended the SMC' algorithm to allow for multiple recombination events between these markers.

The new algorithm is as follows:

1) Simulate a normal coalescent tree without recombination on the left marker. This initial tree has a total size of $T = T_0$ generations.
2) Draw the number *z* of recombination events to occur between the two markers distant of *r* recombination units as $z \sim \text{Poisson}\ (\lambda = rT)$.
3) Draw *z* random numbers uniformly distributed between 1 and *T* to locate the positions of the *z* recombination events on the tree.
4) Locate the position of the most recent recombination event, and color in blue the branch above this event. All recombination events occurring on a blue branch are temporarily deactivated.
5) Detach the recombination event, and let it evolve until the time of the next active recombination event.
6) If the recombining lineage coalesces with a blue lineage, the lineage is colored back in red and the recombination events above this coalescent event become active again.
7) The next active recombination event is implemented, and we have a new recombining lineage free to evolve and to coalesce either with lineages on the tree or with other detached recombining lineages.
8) Steps 6-7 are repeated until only one lineage remains.
9) Remove all remaining blue lineages to keep a fully red tree.

This algorithm is illustrated below

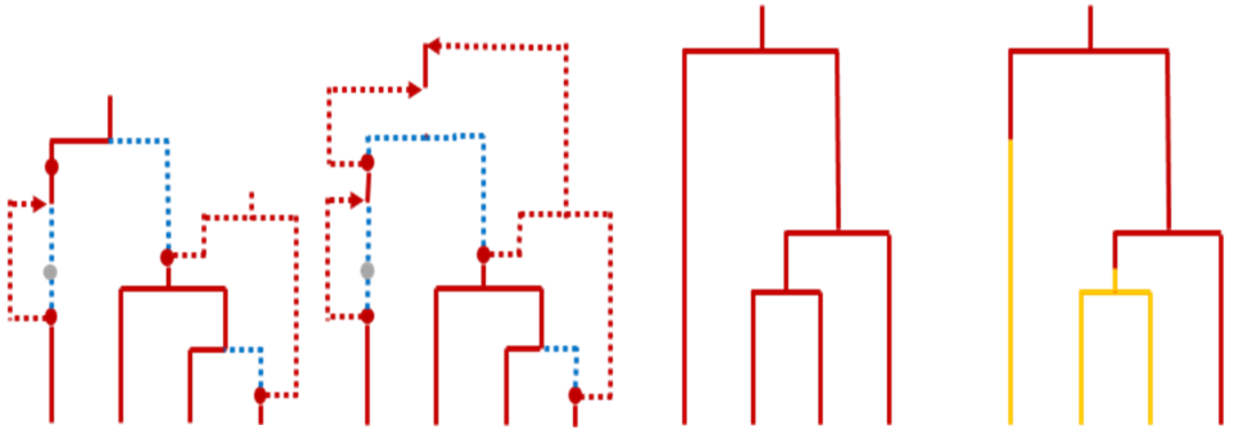


The *z* recombination events are randomly positioned on the tree

The first recombinant lineage is detached and evolves freely backward in time

The recombination event on blue branches are deactivated (grayed)

Two recombinant lineages can coalesce



A recombinant lineage coalesces with a blue branch, which reactivates the recombination events above it.

The final coalescent event is reached.

All blue branches are erased to produce the final tree for the right marker.

The shared lineages between the two trees are shown in yellow.

INTEGRATION INTO APPROXIMATE BAYESIAN COMPUTATIONS (ABC)

fastsimcoal2 can easily be integrated into an ABC framework (see e.g. Beaumont et al. 2002, Csillery et al. 2010), using for instance the *ABCToolBox* framework (Wegmann et al. 2010), which was specially developed to handle simcoal2, and which should be therefore able to accommodate *fastsimcoal2* without modification.

ABCToolBox is available on http://cmpg.unibe.ch/software/abctoolbox/ and its manual on http://cmpg.unibe.ch/software/abctoolbox/ABCtoolbox_manual.pdf.

An alternative, would be to use *fastsimcoal2* built-in parameter sampler to generate *params* files and *Arlequin* arp files , which could then be processed by *arlsumstat* (available on http://cmpg.unibe.ch/software/arlequin35/) to produce summary statistics. A file combining these parameters and their associated summary statistics could then be used for ABC estimation, using different software, like

- ABCest3 (http://cmpg.unibe.ch/software/ABC/) made by L. Excoffier
- ABCReg (Thornton 2009) http://www.molpopgen.org/software.html
- ABCEstimator in ABCToolBox (Wegmann et al. 2010) http://cmpg.unibe.ch/software/abctoolbox/

As mentioned above, an alternative use of *fastsimcoal2* is to use the site frequency spectrum to estimate parameters, which is explained in the next section

ESTIMATION OF DEMOGRAPHIC PARAMETERS FROM THE SFS VIA LIKELIHOOD MAXIMIZATION

We provide below some theoretical foundations for the estimation of demographic parameters from observed (joint) SFS. See Excoffier et al. (2013) for more details.

SIMULATION-BASED LIKELIHOODS

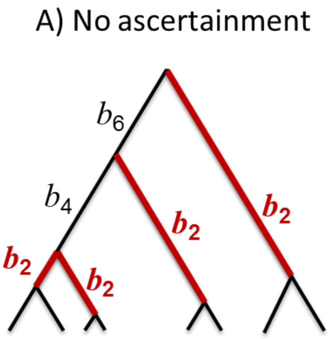
Nielsen (2000)has shown that one could estimate the likelihood of a demographic model $L(X,\theta)$, where *X* is the site frequency spectrum, on the basis of coalescent simulations. This is because the probability p_i of a given derived allele frequency *i* is simply a ratio of branch lengths of the coalescent tree expected under model θ as (Nielsen 2000):

$$p_i = E(t_i \,|\, \theta) \,/\, E(T \,|\, \theta)\,, \tag{1}$$

where t_i is the total length of a set $B_i = \{b_{ij}\}$ of branches directly leading to *i* terminal nodes, and *T* is the total tree length. This probability can then be estimated with arbitrary precision on the basis of Z simulations as

$$\hat{p}_i = \sum_k^Z \sum_{j \in B_i} b_{ijk} \Big/ \sum_k^Z T_k \;. \tag{2}$$

where b_{ijk} is the length of the *j*-th compatible branch in simulation *k* (see Figure below). Note that the estimator shown in eq. 2 implicitly weights simulations according to the probability that a mutation occurs on the simulated tree. This approach is appropriate for analyzing SNPs extracted from DNA sequence data as it is unlikely that SNPs will occur in genomic regions with shallow trees. Note that an estimator of the form $\hat{p}_i = \sum_k^Z \sum_{j \in B_i} b_{ijk} / T_k$ (as used by Garrigan (2009) to estimate the expected SFS) would give each tree the same weight and would thus give an excessive weight to genomic regions with shallow coalescent trees, which can be a problem for recently bottlenecked populations. If some simulated entries of the SFS were zero (because $\sum_k^Z \sum_{j \in B_i} b_{ijk} = 0$), $\hat{p}_i$ was set to an arbitrarily small values (as in Garrigan 2009) chosen here as $\min(\hat{p}_j \,|\, \hat{p}_j > 0)/100$.


A) No ascertainment

COMPOSITE LIKELIHOODS

Probabilities inferred from the simulations and eq. (2)can then be used to compute the composite likelihood of a given model as (Adams and Hudson 2004)

$$CL = \Pr(X \,|\, \theta) \propto P_0^{L-S}(1-P_0)^S \prod_{i=1}^{n-1} \hat{p}_i^{\,m_i} \;. \tag{3}$$

where $X = \{m_1, \ldots, m_{n-1}\}$ is the SFS in a single population sample of size *n*, *S* is the number of polymorphic sites, *L* is the length of the studied sequence, and P_0 is the probability of no mutation on the tree, obtained as $P_0 = e^{-\mu T}$ assuming a Poisson distribution of mutations occurring at rate μ. This formulation can be extended for the joint SFS of two populations as

$$CL_{12} \propto P_0^{L-S}(1-P_0)^S \prod_{i=0}^{n_1} \prod_{j=0}^{n_2} \hat{p}_{ij}^{\,m_{ij}}\,, \tag{4}$$

and one can define a *v*-dimensional SFS for more than two (*v*) populations as

$$CL_{1\ldots v} \propto P_0^{L-S}(1-P_0)^S \prod_{i_1}^{n_1}\prod_{i_2}^{n_2}\cdots\prod_{i_v}^{n_v} \hat{p}_{\Phi}^{\;m_{\Phi}} \tag{5}$$

where $\Phi = i_1\, i_2 \cdots i_{v-1}\, i_v$ is a composite index. However, when the number of populations in the model is larger than 2 and samples sizes are relatively large, the number of entries in the v-dimensional SFS can be huge, implying that most entries of the observed SFS will be either zero or a very small number and that the expected values for these low-count entries will be difficult to estimate precisely. In that case, we have chosen to estimate the *v*-dimensional $CL_{1\ldots v}$ by collapsing all entries with observed SFS less than a predefined threshold ε as

$$CL_{1\ldots v} \propto P_0^{L-S}(1-P_0)^S \left(\prod_{obsSFS_i \geq \varepsilon} \hat{p}_i^{\;m_i}\right)\left(\sum_{1<obsSFS_j<\varepsilon} \hat{p}_j\right)^{\sum_{1<obsSFS_j<\varepsilon} m_j} . \tag{6}$$

When *v*>4, this approach will also prove computationally difficult, and in that case we have chosen to compute a composite composite-likelihood (*C2L*) obtained by multiplying all pairwise CL's, as

$$C2L_{1\ldots v} \propto \prod_{i<j} CL_{ij} \,, \tag{7}$$

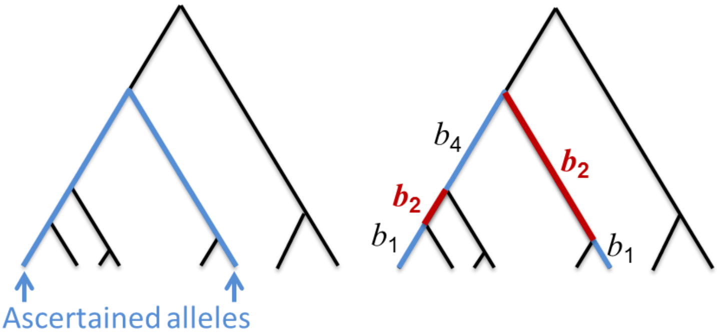
where CL_{ij} is given by eq. (4).

MAXIMIZING THE LIKELIHOOD

As the likelihood is obtained by simulations, which incurs some approximation, we cannot use optimization methods based on partial derivatives. Even though other methods would be possible, we have chosen to use a conditional maximization algorithm (ECM, Meng and Rubin 1993), which is an extension of the EM algorithm where each parameter of the model is maximized in turn, keeping the other parameters at their last estimated value. The maximization of each parameter was done using Brent's (1973, Chapter 5) algorithm, which is a root-finding algorithm using a combination of bisection, secant and inverse quadratic interpolation (see e.g. Press et al. 2007). We start with initial random parameter values, and perform a series of ECM optimization cycles until estimated values stabilize or until we have reached a specified maximum number of ECM cycles (usually 20-40). We advise the use of at least 100,000 coalescent simulations for the estimation of the expected SFS for a given set of demographic parameters.

ESTIMATING DEMOGRAPHIC PARAMETERS FROM SNPS WITH KNOWN ASCERTAINMENT

Recently, Affymetrix developed a new SNP array including ~629,000 SNPs with known ascertainment scheme for population inference (Axiom® Genome-Wide Human Origins 1 Array, http://www.affymetrix.com/support/technical/byproduct.affx?product=Axiom_GW_HuOrigin) (Lu et al. 2011). This array, abbreviated hereafter GWHO, is made up of SNPs defined in 13 discovery panels. In the first 12 panels, SNPs have been identified by comparing the two chromosomes of an individual from a known population, further quality checks and validation on a large population sample (Lu et al. 2011). The 13th panel contains SNPs that are polymorphic when comparing the Denisovan sequence and a random San chromosome. Raw genotypes from 943 unrelated individuals from more than 50 worldwide populations are freely available on ftp://ftp.cephb.fr/hgdp_supp10/.



The ascertainment scheme of this array is simple and homogeneous over a given panel. However, the SFS inferred from this array is biased as only mutations that occur in the ancestry of the two compared chromosomes will be considered (see Figure above). Nielsen et al. (2004) have shown how to correct the expected SFS within a given population under such a simple ascertainment scheme. One could thus be tempted to use their method to unbias the SFS, and then use this unbiased SFS for parameter inference, but this correction method will not work for joint SFS inferred in more than one population. Thus, rather than unbiasing the SFS, we have chosen to infer demographic parameters directly from the ascertained (joint) SFS. It implies we need to model the ascertainment scheme in the coalescent simulations such as to infer the expected ascertained SFS for a given demography. In order to estimate the SFS when SNPs are defined as being sites heterozygous in a given individual, we use the following procedure: 1) we perform conventional coalescent simulations under a given demography, 2) we choose two lineages at random in the ascertained population, 3) we identify the subtree relating the chosen lineages to their most recent common ancestor (MRCA) (highlighted in blue in Figure above), 4) we update the numerator in eq. (2) by summing up branch lengths of the blue subtree that are ancestral to i_1 lineages in population 1, i_2 lineages in population 2, …, i_v lineages in population *v*, 5) The denominator of eq. (2) is updated by summing up the total length of the blue subtree. Parameter optimization is then performed similarly to the unascertained case.

RUNNING FASTSIMCOAL2 ON A CLUSTER

For faster computations *fastsimcoal2* can be launched on a Linux cluster.

SIMULATION OF GENETIC DIVERSITY

As an example, the following bash file `fs_scratch.sh` can be submitted on a cluster running the freely available Sun Grid Engine (SGE, http://gridengine.sunsource.net/).

fs_scratch.sh

```
#!/bin/bash

#get local directory
directory=`pwd`

#$ -cwd
# specify resources needed
#$ -l h_cpu=48:00:00
```

```
# using a scratch directory, reserving disk space and enough files for simulations
#$ -l scratch=1,scratch_size=100M,scratch_files=5k
#$ -N fs_run
#$ -o fs.out
#$ -e fs.err
#$ -m a
#$ -q all.q

#Copying all files to scratch directory
cp * $TMP
cd $TMP

#Running fastsimcoal2
./ fsc25 -t 1PopDNArand.tpl -n 1 -e 1PopDNArand.est -E 1000 -q

#copying results from scratch to original directory
cp * $directory
cd $directory
```

The simple `qsub` command can be used to launch this bash file on a queue as

```
qsub fs_scratch.sh
```

Note that in *fs_scratch.sh*, *fastsimcoal2* is using a scratch directory to write its output. This scratch directory is a temporary directory created by SGE on the node's hard disk, so that *fastsimcoal2* does not use the net to transfer files from the node to the master during simulations, which usually increases speed.

To use the scratch file, one needs to specify how much RAM is needed on the disk (*scratch_size*), and how many files will be created (*scratch_files*). These parameters need to be carefully adjusted. Very long DNA sequences may require several Gb per simulations! Also keep in mind that when using *tpl* and *est* files, (*-n* × *-E*) arp files will be created (*1* × *1000* in `fs_scratch.sh`), as well as *-E* par files, and *-E arb* files. In addition, when very long DNA sequences are generated, with many polymorphic sites, *fastsimcoal2* creates temporary files in a *./garbage* directory, which is then deleted after the *arp* file has been written. In the garbage directory, *fastsimcoal2* will create as many files as the total number of sampled genes (sum of sample sizes) defined in the *par* or the *tpl* file.

Finally, do not forget to make *fastsimcoal2* and your bash file executable on your Linux cluster. This is usually done with the *chmod* command as

```
chmod +x fsc25
chmod +x fs_scratch.sh
```

The use of a scratch is not compulsory and a simpler bash file without use of a scratch directory would simply look like:

`fs.sh`

```
#!/bin/bash
#$ -cwd
# specify resources needed
#$ -l h_cpu=48:00:00
#$ -N fs_run
#$ -o fs.out
#$ -e fs.err
#$ -m a
#$ -q all.q

#Running fastsimcoal2
./ fsc25 -t 1PopDNArand.tpl -n 1 -e 1PopDNArand.est -E 1000 -q
```

Note that a bash file can be created on the fly and qsubmitted from within another bash file. Here is an example, which will submit 10 instances of `fs.h` on 10 different nodes, each one making 1000 simulations from different random parameter values .

```
./launch_fs.sh 10 1PopDNArand 10 1
```

launch_fs.sh

```
#!/bin/bash

#Laurent Excoffier December 2010
#
# The script will launch several instances of fastsimcoal2

#Creating a shortcut for fastsimcoal2
fs= fsc25

if [ $# -ne 4 ]; then
   echo "Expecting the following values on the command line, in that order"
   echo "   Number of instances to run"
   echo "   Generic name of template file"
   echo "   Number of random parameter to draw"
   echo "   Number of simulations per sets of parameter values"
else
   #Using values from the command line
   numInstances=$1
   genericName=$2
   numEstimates=$3
   numSimsPerEst=$4
   echo "numInstances=$numInstances"
   echo "genericName=$genericName"
   echo "numEstimates=$numEstimates"
   echo "numSimsPerEst=$numSimsPerEst"
fi

#Directory for job console outputs
msgs=consoleOutputs
mkdir $msgs 2>/dev/null

echo "Launching ${numInstances} instances of $fs"
let COUNT=numInstances
instancesLaunched=0
while [ $COUNT -gt 0 ]; do
  curInst=fs_job${COUNT}.sh
  newDir=${genericName}_res${COUNT}
  mkdir ${newDir} 2>/dev/null
  cp ./$fs ${newDir}/.
  cp ./${genericName}.est ${newDir}/.
  cp ./${genericName}.tpl ${newDir}/.
  cp ./$fs  ${newDir}/.
  cd  ${newDir}
  let instancesLaunched=instancesLaunched+1
  if [ -e ./$fs ] ; then
     (
    echo "#!/bin/bash"
    echo "#$ -cwd"
    echo "# specify resources needed"
    echo "#$ -l h_cpu=48:00:00"
    echo "#$ -N fs${COUNT}_run"
    echo "#$ -o ../$msgs/fs${COUNT}.out"
    echo "#$ -e ../$msgs/fs${COUNT}.err"
    echo "#$ -m a"
    echo "#$ -q all.q"
    echo ""
    echo "chmod +x ./$fs"
    echo "./$fs -t ${genericName}.tpl -n${numSimsPerEst} -e ${genericName}.est -E${numEstimates} -q"
    echo "rm ./$fs"
    ) > $curInst
    chmod +x $curInst
    qsub ${curInst}
  else
    echo "File $fs not found. Aborting instance $instancesLaunched"
  fi
  cd ..
  let COUNT=COUNT-1
done
```

ESTIMATION OF DEMOGRAPHIC PARAMETERS FROM THE SFS

Since the estimation of demographic parameters from the (joint) SFS requires the maximization of the likelihood of a model, it is necessary to repeat this estimation process several times and to get the global maximum likelihood solution. This is best done by launching several estimations from the same data set on a cluster.

The following bash file does just that, assuming that the observed data (*.obs files) are in a directory called 1PopBot20Mb

It will launch 50 instances of *fastsimcoal2* in 50 different directories (run1 to run50).

This bash file is pretty generic and could be easily modified for any other demographic model, as well as for estimations from ascertainment SNP chip (if with Ascertainment=1)

Again it assumes that there the Sun Grid Engine running on a cluster, and that there is a queue called "all.q" where jobs are going to be submitted.

launchFastSimCoal_1PopBot20Mb.sh

```
#!/bin/bash

#Laurent Excoffier February 2013
#
# The script will launch several jobs of fastsimcoal2 to estimate
# demographic parameters from the SFS, using a each time using a
# conditional maximization (ECM) of the parameter likelihood
# This should run on any kind of SFS files generated by fastsimcoal2
#
# It assumes the following structure of the observed sfs files:
#     scriptDir
#     |
#     |- - - *.est file
#     |- - - *.tpl file
#     |- - - fsc25
#     |- - - targetDir
#                  |
#                  - *.obs files
#

fsc= fsc25
jobcount=0
msgs=conOutputs

#-------- Number of different runs per data set ------
numRuns=50
runBase=1
#-----------------------------

mkdir $msgs 2>/dev/null

#-------- Default run values ------
iniNumSims=100000                 #-n command line option
maxNumSims=100000                 #-N command line option
minNumCycles=10                   #-l command line option
maxNumCycles=40                   #-L command line option
stopCrit=0.001                    #-M command line option
minValidSFSEntry=1                #-C command line option

#-------- Ascertainment ------
withAscertainment=0
ascPop=0                          #-a command line option
ascSize=2                         #-A command line option
#-----------------------------
useMonoSites=""
#useMonoSites="-0"                #-0 command line option
#---------multiSF------------
multiSFS=""
#multiSFS="--multiSFS"            #--multiSFS command line option
#-----------------------------

#-------- Generic Name ------
genericName=1PopBot20Mb
tplGenericName=1PopBot20Mb
estGenericName=1PopBot20Mb
#-----------------------------
```

```
for dirs in $genericName
do
      #Check that dirs is a directory
      if [ -d "$dirs" ]; then
            cd $dirs
                   echo "Main directory : $dirs"
                   estFile=$estGenericName.est
                   tplFile=$tplGenericName.tpl
                   for (( runsDone=$runBase; runsDone<=$numRuns; runsDone++ ))
                   do
                         runDir="run$runsDone"
                         mkdir $runDir 2>/dev/null
                         echo "--------------------------------------------------------------------"
                         echo ""
                         echo "Currrent file: $subDirs $runDir"
                         echo ""
                         cd $runDir
                         #Copying necessary files
                         cp ../../$fsc .
                         cp ../../$tplFile .
                         cp ../../$estFile .
                         cp ../*.obs .
                         #Renaming files for consistency
                         mv $tplFile ${genericName}.tpl
                         mv $estFile ${genericName}.est
                         let jobcount=jobcount+1
                         jobName=${genericName}${jobcount}.sh
                         #Creating bash file on the fly
                         (
                         echo "#!/bin/bash"
                         echo ""
                         echo "#$ -cwd"
                         echo ""
                         echo "# specify resources needed"
                         echo "#$ -l h_cpu=500:00:00"
                         echo ""
                         echo "#$ -N j1P_${jobcount}"
                         echo "#$ -o ../../../$msgs/1P_$runsDone.out"
                         echo "#$ -e ../../../$msgs/1P_$runsDone.err"
                         echo "#$ -m a"
                         echo "#$ -q all.q"
                         echo ""
                         echo "#chmod +x ./$fsc"
                         echo ""
                         echo "#Computing likelihood of the parameters using the ECM-Brent algorithm"
                         echo "echo \"\""
                         if [ $withAscertainment -eq 1 ] ; then
                               echo "./$fsc -t ${genericName}.tpl -n $iniNumSims -N$maxNumSims -d -e
                               ${genericName}.est -M $stopCrit -l$minNumCycles -L $maxNumCycles
                               -a${ascPop} -A${ascSize} -q ${useMonoSites} ${multiSFS}"
                         else
                               echo "./$fsc -t ${genericName}.tpl -n $iniNumSims -N$maxNumSims -d -e
                               ${genericName}.est -M $stopCrit -l$minNumCycles -L $maxNumCycles -q
                               ${useMonoSites} ${multiSFS}"
                         fi
                         echo ""
                         echo "echo \"\""
                         echo "rm ./$fsc"
                         echo "echo \"Job $jobcount terminated\""
                         ) > $jobName
                         chmod +x $jobName
                         echo "Bash file $jobName created"
                         qsub ./${jobName}
                         #./${jobName}
                         cd .. #$runDir
                   done
            cd .. #dirs
      fi
done
```

COMPARATIVE SPEED TESTS: *FASTSIMCOAL* VS. *MS* AND *MACS*

We report below speed tests comparing *fastsimcoal* to *ms* and *MaCS* running all on a Linux cluster made up of 2.6GHz AMD Opterons with 4 GB of RAM and 74 GB HD.

DATA SETS

The following test data sets were used in our comparisons.

	No. of populations	Diploid population size	Migration rate (gen^{-1})	Mutation rate (gen^{-1} × bp^{-1})	Recombination rate (gen^{-1} × bp^{-1})	Sample size per population (no. of genes)
1popNoRec	1	12500		2×10^{-8}	0	2000
1popSmallSample	1	12500		2×10^{-8}	1.2×10^{-8}	20
1popLargeSample	1	12500		2×10^{-8}	1.2×10^{-8}	2000
2popNoRec	2	6250	0.001	2×10^{-8}	0	1000
2popSmallSample	2	6250	0.001	2×10^{-8}	1.2×10^{-8}	10
2popLargeSample	2	6250	0.001	2×10^{-8}	1.2×10^{-8}	1000

The population, mutation and recombination parameters correspond to those used by Chen *et al*. (2009), in their comparison of *ms* to *MaCS* in the one population case.

RESULTS

n=2000, no recombination (CPU times in seconds)

Data set	No. of replicates	Sequence length	Program		
			ms	*MaCS*	*fastsimcoal*
1popNoRec	1000	1 Mb	1.1	11.1	9.5
	100	10 Mb	9.6	107.0	72.9
	100	100 Mb	147.9	1319.5	1038.1
2popNoRec	1000	1 Mb	1.2	12.5	9.3
	100	10 Mb	8.9	128.1	71.5
	100	100 Mb	161.2	1513.2	1099.9

Without recombination, *ms* is much faster than the two other programs based on the SMC' approximation, and *fastsimcoal* becomes increasingly faster than *MaCS* with larger recombination rates and with migration.

n=20, recombination (CPU times in seconds)

Data set	No. of replicates	Sequence length	Program		
			ms	*MaCS*	*fastsimcoal*
	1000	1 Mb	0.344	0.242	0.095
1popSmallSample	100	10 Mb	159.246	2.618	0.460
	100	100 Mb	x	26.124	4.364
	1000	1 Mb	0.378	0.907	0.152
2popSmallSample	100	10 Mb	165.507	9.094	1.080
	100	100 Mb	x	97.876	10.559

x : *ms* crashed

For small sample sizes (total *n*=20) and with recombination, the SMC' based programs are becoming much faster than *ms*, which fails to run for 100Mb sequences. For such small sample sizes, *fastsimcoal* is 2.5 to 9.3 times faster than *MaCS*. For *MaCS* and *fastsimcoal*, computing time increases approximately linearly with sequence length, as expected.

n=2000, recombination (CPU times in seconds)

Data set	No. of replicates	Sequence length	Program		
			ms	*MaCS*	*fastsimcoal*
	1000	1 Mb	3.7	28.1	25.2
1popLargeSample	100	10 Mb	x	327.5	235.7
	100	100 Mb	x	3700.8	2635.4
	1000	1 Mb	3.9	33.3	25.9
2popLargeSample	100	10 Mb	x	393.5	240.6
	100	100 Mb	x	4311.1	2684.7

x : *ms* crashed

For large sample sizes (total *n*=2000), *ms* is actually faster than the two other programs for a "small" sequence of 1Mb, but failed to run successfully for longer sequences. For these large sample sizes, *fastsimcoal* is 1.2 to 1.8 times faster than *MaCs*. *fastsimcoal* computing time still increases approximately linearly with sequence length, which is not the case of *MaCS*, which becomes slightly penalized by larger sequences. Note however, that for 1Mb and 10Mb, we used *fastsimcoal* `-k` options allowing it to keep all simulated sites in memory before writing them to the output file, which was not possible for 100Mb sequences, which would use up too much memory.

COMPARATIVE SPEED TESTS: FASTSIMCOAL21 VS. FASTSIMCOAL2.5 (FSC25)

We report here some comparative speed test done under Win8.1, using the 64 bit version of fastsimcol21 and fsc25. The tests were performed on a DELL computer equipped with an Intel i7-3770 processor at 3.4 Ghz (quadcore).

	Programs				
		fsc25 (1 thread)		fsc25 (8 threads)	
Data set	**fastsimcoal21**	**-c1 -B1**	**gain**	**-c8 -B8**	**gain**
DNA sequence					
constant population of 10,000 diploids, n=100)					
10 x 100000 x 100 bp no rec	18.08 s	12.65 s	1.43	3.85 s	4.70
10 x 1000000 x 100 bp no rec	26.04 s	22.97 s	1.13	8.50 s	3.06
10 x 10 Mb with rec	10.47 s	10.34 s	1.01	10.37 s	1.01
1 x 100 Mb with rec	13.59 s	12.88 s	1.06	13.62 s	1.00
FREQ (parameter estimation)					
1PopExpInst20Mb -n100000 -l5	14.53 s	8.14 s	1.79	2.53 s	5.74
2PopDiv20Mb -n100000 -l5	27.77 s	20.91 s	1.33	4.91 s	5.66
2PopDivMigr20Mb -n100000 -l5	48.94 s	35.54 s	1.38	8.45 s	5.79
3PopExpBotm -n10000 -l5	200.48 s	167.66 s	1.20	35.63 s	5.63
3PopExpBotm mSFS -n10000 -l5	123.39 s	111.96 s	1.10	24.00 s	5.14
10Pop2ContiIsl 1 -n1000 -l5	191.11 s	129.35 s	1.48	32.54 s	5.87
10Pop2ContiIsl 2 -n1000 -l5	182.18 s	121.07 s	1.50	32.38 s	5.63

Optimizations have not been performed in case of recombination, and in that case *fsc25* performs in a way similar to *fastsimcoal21*. In absence of recombination, and with a single thread activated (as on a cluster), speed gain can be of up to 43% for DNA sequence simulation, and up to 80% for parameter estimation. When using the multithreaded option (-c0 using all available computing threads, 8 in the present case), the speed gain is 3 to 4x for DNA sequence simulation, and > 5x for parameter estimation. Again, multithreading was not implemented for runs with recombination, but there is no penalty activating this option.

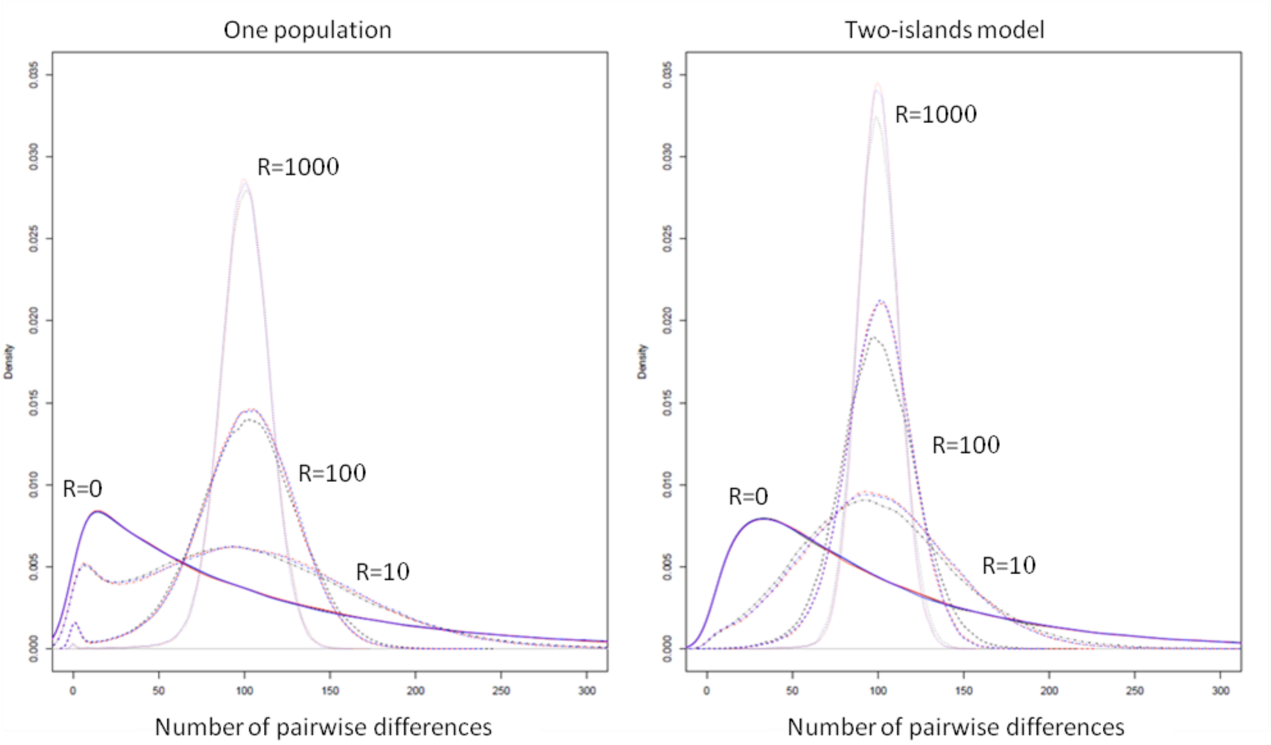
Note that the command line options are not optimal for parameter estimation, but they were just chosen for achieving reasonable computing times for comparative purposes.

The bash file “*fsc_speed_test.sh*” and all necessary files to perform these tests can be found in the “example file” directory.

COMPARATIVE PATTERNS OF SIMULATED MOLECULAR DIVERSITY

NUMBER OF PAIRWISE DIFFERENCES

We report below a comparison of the patterns of diversity within and between populations simulated by *ms*, *MaCs* and *fastsimcoal*.



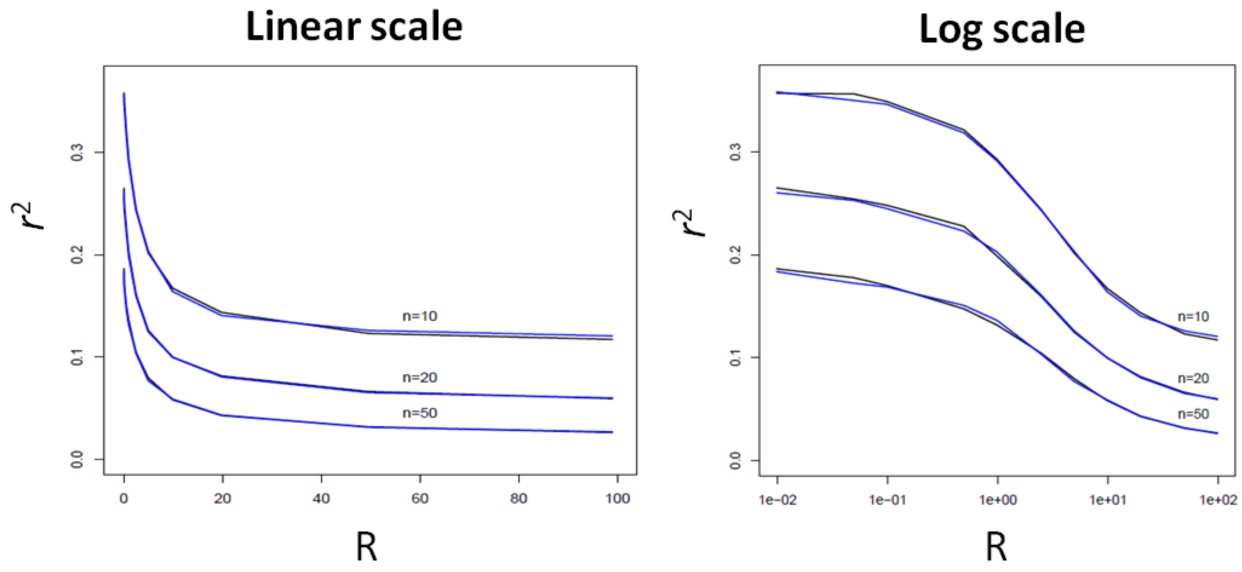
ms results are shown with a black line, *MaCs* with a red line, and *fastsimoal*, with a blue line. In all cases, the empirical distributions of the number of pairwise differences were computed from 100,000 simulations of the coalescent of 2 genes. The 2 genes were drawn from a single population for the one-population case, and were drawn each from a different population in the two-island model case. We used the following population parameters: $\theta = 4N\mu = 100$ for the entire sequence in the one population case; $\theta = 4N_0\mu = 4N_1\mu = 25$ and $M = 4Nm = 0.5$ for the two-island case; and $R = 4Nr$ (where *r* is the total recombination rate between the two ends of the sequence to be simulated) was varied between 0 and 1000, as shown above.

In all cases, *MaCS* and *fastsimcoal* lead to identical distributions, which is expected as they are both based on the same SMC' approximation. In absence of recombination *MaCS* and *fastsimcoal* give also exactly the same distributions as *ms*, but are just running 7-10 times slower, as was seen in the previous section. With very high recombination rates, the SMC-based approximation of *MaCS* and *fastsimcoal* is extremely close to the ancestral recombination graph (ARG) implemented in *ms*, in keeping with previous results (McVean and Cardin 2005). For "intermediate" recombination rates (*R*=10, *R*=100), some slight differences do emerge between ARG- and SMC-based programs, and these differences are slightly more pronounced in the 2-island model. However, it seems that these differences are much less than differences due to the choice of different demographic, mutation, or recombination parameters.

LINKAGE DISEQUILIBRIUM

Previous studies have shown that patterns of LD were virtually undistinguishable between *ms* and SMC'-based programs (Marjoram and Wall 2006, Chen et al. 2009) along DNA sequences.

Here, we report a comparison of the average LD (as measured by r^2) between *simcoal2* and *fastsimcoal* between two SNP markers located at a given recombination distance expressed in *R* units.



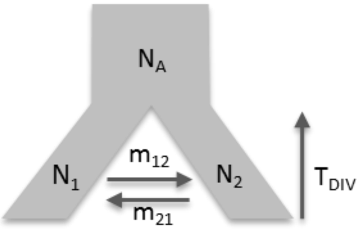
Linear scale

Log scale

The results are based on 20,000 simulations and confirm that average r^2 values are virtually undistinguishable between the two approaches.

EXAMPLE FILES FOR THE ESTIMATION OF DEMOGRAPHY FROM THE (JOINT) SFS

ISOLATION WITH MIGRATION (IM) SCENARIO



IM20Mb.tpl

```
//Parameters for the coalescence simulation program : simcoal.exe
2 samples to simulate :
//Population effective sizes (number of genes)
NPOP1
NPOP2
//Samples sizes and samples age
20
30
//Growth rates: negative growth implies population expansion
0
0
//Number of migration matrices : 0 implies no migration between demes
2
//Migration matrix 0
0 MIG21
MIG12 0
//Migration matrix 1
0 0
0 0
//historical event: time, source, sink, migrants, new deme size, growth rate, migr mat index
1 historical event
TDIV 0 1 1 RESIZE 0 1
//Number of independent loci [chromosome]
1 0
//Per chromosome: Number of contiguous linkage Block: a block is a set of contiguous loci
1
//per Block:data type, number of loci, per gen recomb and mut rates
FREQ 1 0 2.5e-8e-8
```

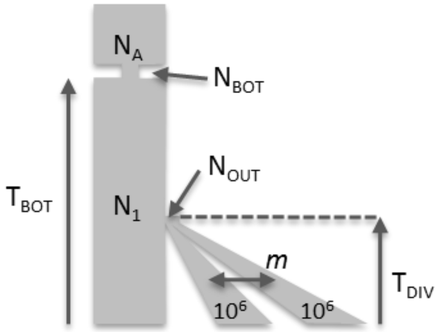
IM20Mb.est

```
// Priors and rules file
// *********************

[PARAMETERS]
//#isInt? #name   #dist.#min  #max
//all N are in number of haploid individuals
1  ANCSIZE     unif     100  100000   output
1  NPOP1       unif     100  100000   output
1  NPOP2       unif     100  100000   output
0  N1M21       logunif  1e-2 20       hide
0  N2M12       logunif  1e-2 20       hide
1  TDIV        unif     100   20000   output

[RULES]

[COMPLEX PARAMETERS]

0  RESIZE = ANCSIZE/NPOP2     hide
0  MIG21  = N1M21/NPOP1       output
0  MIG12  = N2M12/NPOP2       output
```

COMMAND LINE FOR PARAMETER ESTIMATION

```
./ fsc25 -t IM20Mb.tpl -n100000 -N100000 -d -e IM20Mb.est -M 0.001 -l 10 -L 40 -c6 -q
```

DIVERGENCE OF THREE POPULATIONS



3PopExpBot20Mb.tpl

```
//Parameters for the coalescence simulation program : fastsimcoal.exe
3 samples to simulate :
//Population effective sizes (number of genes)
NPOPAF
2000000
2000000
//Samples sizes and samples age
20
20
20
//Growth rates : negative growth implies population expansion
0
R1
R1
//Number of migration matrices : 0 implies no migration between demes
2
//Migration matrix 0
0.0000 0.0000 0.0000
0.0000 0.0000 MIG
0.0000 MIG 0.0000
//Migration matrix 1
0 0 0
0 0 0
0 0 0
//historical event: time, source, sink, migrants, new deme size, growth rate, migr mat index
4 historical event
TDIV 2 0 1 1 0 1
TDIV 1 0 1 1 0 1
TBOT 0 0 0 RES1 0 1
TENDBOT 0 0 0 RES2 0 1
//Number of independent loci [chromosome]
1 0
//Per chromosome: Number of contiguous linkage Block: a block is a set of contiguous loci
1
//per Block:data type, number of loci, per gen recomb and mut rates
FREQ 1 0 2.5e-8
```

3PopExpBot20Mb.est

```
// Priors and rules file
// *********************

[PARAMETERS]
//#isInt? #name   #dist.#min  #max
//all Ns are in number of haploid individuals
1  ANCSIZE     unif     1000  100000  output
1  NBOT        unif     10    2000    output
1  NPOPAF      unif     1000 100000   output
1  NPOPOOA     unif     10    10000   output
1  TDIV        unif     10    10000   output
1  TPLUSDIV    unif     10    10000   hide
0  MIG         logunif  1e-5 1e-2     output
```

```
[RULES]

[COMPLEX PARAMETERS]
1  TBOT = TDIV+TPLUSDIV            output
0  RATIO_OOA_EA = NPOPOOA/2000000 hide
0  RTEA = log(RATIO_OOA_EA)       hide
0  R1 = RTEA/TDIV                 hide
1  TENDBOT = TBOT+500             hide
0  RES1 = NBOT/NPOPAF             hide
0  RES2 = ANCSIZE/NBOT            hide
```

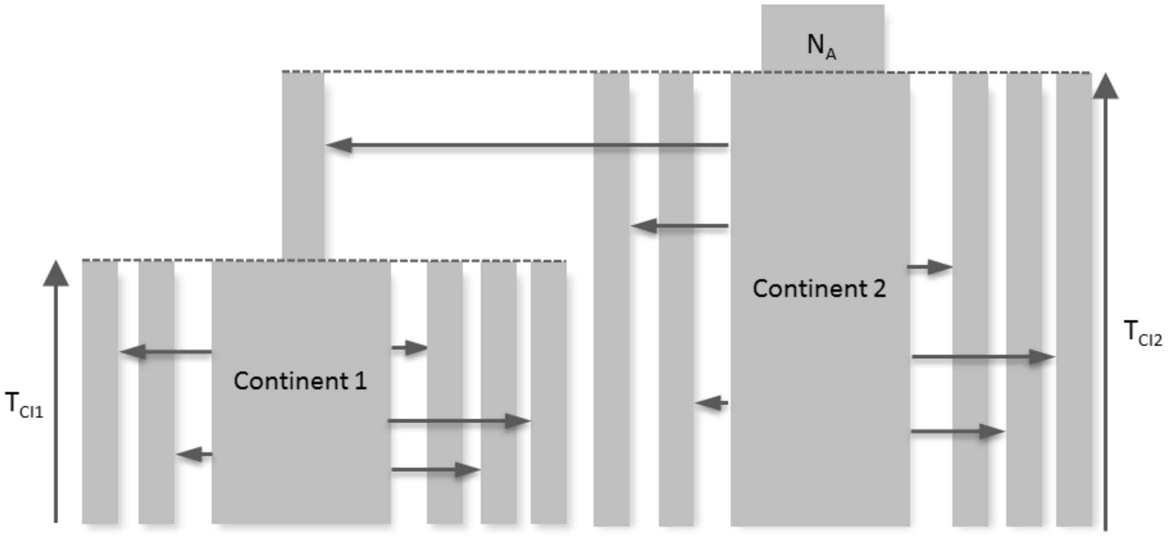
COMMAND LINE FOR PARAMETER ESTIMATION

```
./ fsc25 -t 3PopExpBot20Mb.tpl -n100000 -N100000 -d -e 3PopExpBot20Mb.est -M 0.001 -l 10 -L 40 -q --
multiSFS -C10 -c8
```

Comments:

- Here we use the multidimensional SFS (`--multiSFS`)
- We also impose a minimum observed SFS entry of 10 (`-C10`) to appear in the likelihood, entries of the observed SFS with lower observations wil be collapsed into a single entry
- We use 8 threads to speed up computations (`-c8`)

HIERARCHICAL ISLAND MODEL



10Pop2ContiIsl.tpl

```
//Parameters for the coalescence simulation program : fastsimcoal.exe
12 samples to simulate :
//Population effective sizes (number of genes)
1000
1000
1000
1000
1000
1000
1000
1000
1000
1000
20000000
20000000
//Samples sizes and samples age
20
20
20
20
20
20
20
20
20
20
```

```
0
0
//Growth rates      : negative growth implies population expansion
0
0
0
0
0
0
0
0
0
0
0
0
//Number of migration matrices : 0 implies no migration between demes
3
//Migration matrix 0
0         0         0         0         0         0         0         0         0         0         M010      0
0         0         0         0         0         0         0         0         0         0         M110      0
0         0         0         0         0         0         0         0         0         0         M210      0
0         0         0         0         0         0         0         0         0         0         M310      0
0         0         0         0         0         0         0         0         0         0         M410      0
0         0         0         0         0         0         0         0         0         0         0         M511
0         0         0         0         0         0         0         0         0         0         0         M611
0         0         0         0         0         0         0         0         0         0         0         M711
0         0         0         0         0         0         0         0         0         0         0         M811
0         0         0         0         0         0         0         0         0         0         0         M911
0         0         0         0         0         0         0         0         0         0         0         0
0         0         0         0         0         0         0         0         0         0         0         0
//Migration matrix 1
0         0         0         0         0         0         0         0         0         0         0         0
0         0         0         0         0         0         0         0         0         0         0         0
0         0         0         0         0         0         0         0         0         0         0         0
0         0         0         0         0         0         0         0         0         0         0         0
0         0         0         0         0         0         0         0         0         0         0         0
0         0         0         0         0         0         0         0         0         0         0         M511
0         0         0         0         0         0         0         0         0         0         0         M611
0         0         0         0         0         0         0         0         0         0         0         M711
0         0         0         0         0         0         0         0         0         0         0         M811
0         0         0         0         0         0         0         0         0         0         0         M911
0         0         0         0         0         0         0         0         0         0         0         M1211
0         0         0         0         0         0         0         0         0         0         0         0
//Migration matrix 2
0         0         0         0         0         0         0         0         0         0         0         0
0         0         0         0         0         0         0         0         0         0         0         0
0         0         0         0         0         0         0         0         0         0         0         0
0         0         0         0         0         0         0         0         0         0         0         0
0         0         0         0         0         0         0         0         0         0         0         0
0         0         0         0         0         0         0         0         0         0         0         0
0         0         0         0         0         0         0         0         0         0         0         0
0         0         0         0         0         0         0         0         0         0         0         0
0         0         0         0         0         0         0         0         0         0         0         0
0         0         0         0         0         0         0         0         0         0         0         0
0         0         0         0         0         0         0         0         0         0         0         0
0         0         0         0         0         0         0         0         0         0         0         0
//historical event: time, source, sink, migrants, new deme size, new growth rate, migration matrix index
13 historical event
TISLAND1 0 10 1 1 0 1
TISLAND1 1 10 1 1 0 1
TISLAND1 2 10 1 1 0 1
TISLAND1 3 10 1 1 0 1
TISLAND1 4 10 1 1 0 1
TISLAND1 10 10 0 0.0001 0 1
TISLAND2 5 11 1 1 0 2
TISLAND2 6 11 1 1 0 2
TISLAND2 7 11 1 1 0 2
TISLAND2 8 11 1 1 0 2
TISLAND2 9 11 1 1 0 2
TISLAND2 10 11 1 1 0 2
TISLAND2 11 11 0 RESIZE 0 2//Number of independent loci [chromosome]
1 0
//Per chromosome: Number of contiguous linkage Block: a block is a set of contiguous loci
1
//per Block:data type, number of loci, per gen recomb and mut rates
FREQ 1 0 2.5e-8
```

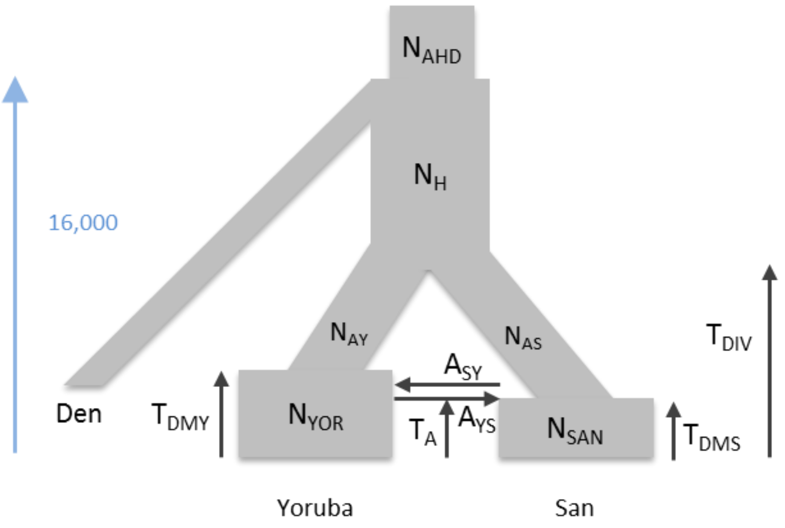
10Pop2ContiIsl.est

```
// Priors and rules file
// *********************

[PARAMETERS]
//#isInt? #name   #dist.#min  #max
//all Ns are in number of haploid individuals
1  ANCSIZE     unif     10  100000    output
0  NM0         logunif  0.01  100     output
0  NM1         logunif  0.01  100     output
0  NM2         logunif  0.01  100     output
0  NM3         logunif  0.01  100     output
0  NM4         logunif  0.01  100     output
0  NM5         logunif  0.01  100     output
0  NM6         logunif  0.01  100     output
0  NM7         logunif  0.01  100     output
0  NM8         logunif  0.01  100     output
0  NM9         logunif  0.01  100     output
0  NM_12        logunif  0.01  100    output
1  TISLAND1    unif     10    20000   output
1  TPLUS       unif     10    20000   hide

[RULES]

[COMPLEX PARAMETERS]

//Assume an island haploid population size of 1000 for all islands
1  TISLAND2 = TISLAND1+TPLUS output
0  M010 =  NM0/1000          hide
0  M110 =  NM1/1000          hide
0  M210 =  NM2/1000          hide
0  M310 =  NM3/1000          hide
0  M410 =  NM4/1000          hide
0  M511 =  NM5/1000          hide
0  M611 =  NM6/1000          hide
0  M711 =  NM7/1000          hide
0  M811 =  NM8/1000          hide
0  M911 =  NM9/1000          hide
0  M1211 = NM_12/2000        hide
0  RESIZE = ANCSIZE/20000000 hide
```

COMMAND LINE FOR PARAMETER ESTIMATION

```
./ fsc25 -t 10Pop2ContiIsl.tpl -n50000 -N50000 -d -e 10Pop2ContiIsl.est -M 0.001 - l10 -L 30 -q
```

HUMAN AFRICAN DEMOGRAPHY WITH SNP ASCERTAINMENT



p4NocpgSanYor4.tpl

```
//Parameters for the coalescence simulation program : simcoal.exe
3 samples to simulate : Exponential growth : 1000 to 100,000,000 started 3000 generations ago
//Population effective sizes (number of genes)
NSan
NYor
1000
//Samples sizes
12
44
1 1600
//Growth rates : negative growth implies population expansion
0
0
0
//Number of migration matrices : 0 implies no migration between demes
0
//historical event: time, source, sink, migrants, new deme size, new growth rate, migration matrix index
6  historical event
TDMS 0 0 0 RES_SAN 0 0
TDMY 1 1 0 RES_YOR 0 0
TAdm 0 1 AYS 1 0 0
TAdm 1 0 ASY 1 0 0
TDIVSanYor 1 0 1 RES_AF 0 0
16000 2 0 1 RES_ANC 0 0
//Number of independent loci [chromosome]
1 0
//Per chromosome: Number of contiguous linkage Block: a block is a set of contiguous loci
1
//per Block:data type, number of loci, per gen recomb and mut rates
FREQ 1 0 1e-7
```

Note that the divergence time with Denisovans is fixed, and this time is used to calibrate the other parameters

p4NocpgSanYor4.tpl

```
// Priors and rules file
// *********************

[PARAMETERS]
//#isInt? #name   #dist.#min  #max
//all Ns are in number of haploid individuals
1  NSan        unif     1000   2e6  output
1  NYor        unif     1000   2e6  output
1  NASan       unif     1000   1e5  output
1  NAYor       unif     1000   1e5  output
1  HSIZE       unif     1000   1e5  output
1  ANCSIZE     unif     1000   1e5  output
1  TDMS        unif     10     500  output
1  TDMY        unif     10     500  output
1  TAdm        unif     10     500  output
1  TPlusAdm    unif     1     5000  hide
0  AYS         unif     0      0.2  output
0  ASY         unif     0      0.2  output

[RULES]

[COMPLEX PARAMETERS]
1  TDIVSanYor = TAdm+TPlusAdm   output
0  RES_SAN  = NASan/NSan        hide
0  RES_YOR  = NAYor/NYor        hide
0  RES_AF   = HSIZE/NASan       hide
0  RES_ANC  = ANCSIZE/HSIZE     hide
```

COMMAND LINE FOR PARAMETER ESTIMATION

```
./ fsc25 -t p4NocpgSanYor4.tpl -n100000 -N100000 -d -e p4NocpgSanYor4.est -M 0.001 -l 10 -L 20 -a0 -
A2 -q -0 -C2 -multiSFS
```

Comments:

- -a0 indicates that the ascertainment occurred in population 0 (San)
- -A2 indicates that 2 chromosomes were used to infer the polymorphism status for each SNP
- -0 indicates that one does not use information on the number of monomorphic sites
- --multiSFS indicates that the 3D SFS is used

10. REFERENCES

Adams, A. M. and R. R. Hudson. 2004. Maximum-likelihood estimation of demographic parameters using the frequency spectrum of unlinked single-nucleotide polymorphisms. Genetics **168**:1699-1712.

Beaumont, M. A., W. Zhang, and D. J. Balding. 2002. Approximate Bayesian computation in population genetics. Genetics **162**:2025-2035.

Brent, R. P. 1973. Algorithms for Minimization without Derivatives. Prentice-Hall, Englewood Cliffs, NJ.

Chen, G. K., P. Marjoram, and J. D. Wall. 2009. Fast and flexible simulation of DNA sequence data. Genome Res **19**:136-142.

Csillery, K., M. G. Blum, O. E. Gaggiotti, and O. Francois. 2010. Approximate Bayesian Computation (ABC) in practice. Trends in ecology & evolution **25**:410-418.

Excoffier, L. and H. E. Lischer. 2010. Arlequin suite ver 3.5: a new series of programs to perform population genetics analyses under Linux and Windows. Mol Ecol Resour **10**:564-567.

Excoffier, L. and M. Foll. 2011. fastsimcoal: a continuous-time coalescent simulator of genomic diversity under arbitrarily complex evolutionary scenarios. Bioinformatics 27: 1332-1334.

Excoffier, L., Dupanloup, I., Huerta-Sánchez, E., and M. Foll (2013) Robust demographic inference from genomic and SNP data. PLOS Genetics 9(10):e1003905.

Garrigan, D. 2009. Composite likelihood estimation of demographic parameters. BMC genetics **10**:72.

Laval, G. and L. Excoffier. 2004. SIMCOAL 2.0: a program to simulate genomic diversity over large recombining regions in a subdivided population with a complex history. Bioinformatics **20**:2485-2487.

Lu, Y., N. Patterson, Y. Zhan, S. Mallick, and D. Reich. 2011. Technical design document for a SNP array that is optimized for population genetics.

Marjoram, P. and J. D. Wall. 2006. Fast "coalescent" simulation. BMC genetics **7**:16.

McVean, G. A. and N. J. Cardin. 2005. Approximating the coalescent with recombination. Philos Trans R Soc Lond B Biol Sci **360**:1387-1393.

Meng, X. L. and D. B. Rubin. 1993. Maximum likelihood estimation via the ECM algorithm: A general framework. Biometrika **80**:267-278.

Nielsen, R. 2000. Estimation of population parameters and recombination rates from single nucleotide polymorphisms. Genetics **154**:931-942.

Nielsen, R., M. J. Hubisz, and A. G. Clark. 2004. Reconstituting the Frequency Spectrum of Ascertained Single-Nucleotide Polymorphism Data. Genetics **168**:2373-2382.

Press, W. H., S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery. 2007. Numerical Recipes in C++: The Art of Scientific Computing. 3rd edition. Cambridge University Press, Cambridge.

Thornton, K. R. 2009. Automating approximate Bayesian computation by local linear regression. BMC genetics **10**:35.

Wegmann, D., C. Leuenberger, S. Neuenschwander, and L. Excoffier. 2010. ABCtoolbox: a versatile toolkit for approximate Bayesian computations. BMC Bioinformatics **11**:116.