A pproximate
B lockwise
L ikelihood
E stimation

Version 0.1.x
December 16, 2016

Champak Beeravolu Reddy
champak.br@gmail.com

Contents

1 Introduction

`ABLE` is a program written in C/C++ for the joint inference of arbitrary population histories and the genome-wide recombination rate using data from multiple whole genome sequences or fragmented assemblies (e.g. UCE's, RADSeq, and targeted exomes). The inference results in a Maximum Likelihood Estimate (MLE) of the parameters corresponding to the demographic model of interest along with the recombination parameter. It makes use of the distribution of blockwise SFS (bSFS) patterns which retain information on the variation in genealogies or Ancestral Recombination Graphs (ARGs) spanning short-range linkage blocks across the genome. `ABLE` does not require phased data as the bSFS does not distinguish the sampled lineage in which a mutation has occurred. Like with the SFS, outgroup information can be also be ignored by folding the bSFS. `ABLE` takes advantage of `openmp` parallelization and is tailored for studying the population histories of model as well as non-model species.

This is the documentation accompanying `ABLE` and the current version of the project is freely available from https://github.com/champost/ABLE.

2 Installation

2.1 Linux

It is easiest to build an `ABLE` binary under all flavours of Linux. `ABLE` requires the GNU Compiler Collection (`gcc`) and GNU Make (`make`) for a smooth installation and has been tested using `gcc 4.8.4` and `make 3.81`. It is also useful to have `git` installed for seamless updates of the latest version of `ABLE`. If you do not have `gcc`, `make` or `git`, you can use your OS specific package handling utility.

Under Ubuntu this corresponds to the following in a terminal

```
sudo apt-get install build-essential git
```

Other dependencies such as the GNU Scientific Library (`GSL`) and the Non-Linear Optimization (`NLopt`) library are automatically installed by following the instructions outlined below.

1. Clone the `ABLE` repository. If `git` is not available see the next section (or here).

   ```
   git clone https://github.com/champost/ABLE.git
   ```

2. Change directory

   ```
   cd ABLE
   ```

3. If you are installing `ABLE` for the **first time** you might have to compile the **GSL** and **NLopt** from source. This can take some time as the command below performs a **static installation** of the libraries. You can skip this step if you already have these libraries installed system-wide or if you are simply updating `ABLE` to the latest version (see next section).

   ```
   make deps
   ```

4. Finally, build an `ABLE` binary

   ```
   make clean && make all
   ```

If you want `ABLE` to be accessible from everywhere, such as your data folder, you might want to

```
cp ABLE ~/bin
```

This ensures that you can execute the program by specifying `ABLE ...` instead of `./ABLE ...` from the installation folder. This holds only if `~/bin` exists and is part of your `$PATH` environment variable.

3 Updates

3.1 With Git

As mentioned above, `git` considerably simplifies the process (or headache) of updating `ABLE`. You simply have to change your current working directory to that of `ABLE` and execute the following

```
git pull
make clean && make all
```

3.2 Without Git

If `git` is not available on your machine you can try looking for solutions here. Alternatively, you can download the `ABLE` repository somewhere on your computer as shown below.

```
wget https://github.com/champost/ABLE/archive/master.tar.gz
```

Untar the archive using

```
tar -xzf master.tar.gz
```

and manually compare the contents of the `ABLE-master` directory with your `ABLE` installation and replace the files that have changed in size.

3.3 Updating dependencies

With or without `git`, you will seldom need to recompile the libraries that `ABLE` depends upon and in which case you will have to run the `make deps` command before compiling an `ABLE` binary with `make clean && make all`. This information will be clearly stated in the Release notes and/or the `ABLE` documentation.

4 Configuration

For `ABLE` to execute correctly, you need to specify a command line, some options in a config file and provide a data file. For very long command lines it might be preferable to specify it from within the config file (see 4.2.2).

4.1 Command line options

The command line needs to be in the ***ms*** format. Note that `ABLE` supports only a subset (shown below) of all the available ***ms*** command line options. Please refer to the full ***ms*** documentation for a better understanding of these options.

- `-t` θ
- `-r` ρ **nsites**
- `-G` α
- `-I` npop n1 n2 ... [$4N_0m$]
- `-eG` t α
- `-eg` t i α_i
- `-eN` t x
- `-en` t i x
- `-eM` t x
- `-em` t i j x
- `-ema` t npop M_{11} M_{12} M_{13}... M_{21} ...
- `-es` t i p
- `-ej` t i j

4.1.1 The `tbi` keyword

While ***ms*** provides for the `tbs` option, we introduce the `tbi` keyword which stands for "**to be inferred**" as part of the `ABLE` command line. `tbi` keywords are to be used instead of the values of the parameters of a demographic model

(*i.e.* some floating value) which need to be inferred. All `tbi` keywords need to be suffixed by a number (`tbi1, tbi2,...`). Thus, all occurrences on the command line of the same demographic parameter can be correctly identified.

Below is a typical example of an `ABLE` command line for a simple history describing a single discrete change in population size defined by three parameters (current and ancestral population sizes and the time of size change) which are *to be inferred* (see 4.4.1):

```
./ABLE 4 100000 -t tbi1 -eN tbi2 tbi3 -T config.txt
```

`ABLE` also requires the user to specify a `-T` towards the end of the command line indicating the start of the config filename (if any) specified by the user. If no filename is specified, by default `ABLE` looks for a file named `config.txt`. See also 4.2.2 for specifying the command line directly inside `config.txt`.

4.2 Config file options

Note

- **All options** of the config file are **case sensitive**.
- Successive options can undo previous ones.
- Comments need to be specified as an entire line in `ABLE`. **Unexpected behaviour** can result if you specify keywords/options and a comment on the **same line** *e.g.* `Key val # comment after keyword`

4.2.1 Syntax

Any line in the config file beginning with the hash symbol `#` will be considered as a comment by `ABLE` and ignored. Every keyword along with its corresponding options are to be specified on a separate line. Also, keywords and options are to be separated by a single space. All config file keywords introduced here will be of one or more of these types :

T1 : `Key`

T2 : `Key val` *or* `Key val1 val2 val3 ...`

T3 : `Key SubKey val` *or* `Key SubKey val1 val2 val3...`

Here, the presence/absence of a T1 keyword respectively represents a binary true/-false option whereas it is possible to specify a lot more with a T2/T3 keyword.

4.2.2 Command line within the config file

`ABLE` (command line)
This is a special keyword which allows the user to specify the `ms` styled command line from within the config file instead of the console/terminal. For instance, the command from section 4.1.1 can be specified anywhere in `config.txt` as follows

```
ABLE 4 100000 -t tbi1 -eN tbi2 tbi3
```

The corresponding command line would then be `./ABLE config.txt`

4.2.3 Keywords

- `datafile` (T2)
  Accepts a single value specifying the name and location (relative to the config file) of the file containing the data (see 4.3 for more on the data format).

  ```
  datafile filename
  ```

- `datafile_format` (T2)
  Data can be specified in either of the two formats below.

  Note

  The **default** input file format is `bSFS`.

  - `datafile_format bSFS`
    Each line in this format contains a tag describing the bSFS configuration (joint bSFS if multiple population samples) and its respective frequency of occurrence in the data.

    ```
    (tag1) : prob1
    (tag2) : prob2
    ...
    ```

    For an example of this format please refer to the accompanying Orangutan dataset in the `/data` folder. Further information on the general logic behind the bSFS/jbSFS tags will be provided here later.

  - `datafile_format pseudo_MS`
    This format closely resembles that of simulated samples output by the ***ms*** software and will be the easiest for all users of `ABLE` to provide. A more detailed description of this format can be found in 4.3. A file named `block_SNPs.txt` is created with the number of SNPs per

sequence block providing for a manual cross-check that `ABLE` has correctly accounted for available polymorphism. While tri/quadri-allelic nucleotide positions are ignored they are however listed in `block_SNPs.txt`.

- `allele_type` (T2)
  This option provides additional information to `ABLE` on how to read in the data when it is already in a `pseudo_MS` format. See 4.3 for full examples of both the formats below.

  - `allele_type genotype`
    Valid characters are `A`, `T`, `G` or `C` and missing information is specified by using `N` (see 4.3.1).
  - `allele_type binary`
    Valid characters are `0` or `1` and missing information is specified by using `N` (see 4.3.2).

- `task` (T2)
  This option defines any one of the three principal tasks (or modes) that `ABLE` is meant to undertake. More information can be found in the examples section (4.4).

  - `task exact_bSFS`
    `ABLE` attempts to calculate the **exact bSFS** for a given number of genealogies to be sampled and there is no requirement for a file containing data. Results obtained under this mode can be readily compared with analytical results. A file named `expected_bSFS.txt` is generated with the expected bSFS for a given number of genealogies and a point in parameter space (see 4.4.1).
  - `task conditional_bSFS`
    This mode of `ABLE` calculates the **conditional bSFS** (*i.e.* only configurations found in the data) at a given point in parameter space and for some number of genealogies (see 4.4.2).
  - `task infer`
    This is the standard mode for inferring demographic parameters and will be the most used feature of `ABLE`. This mode typically consists of a global search followed by a local search and finally a refined log-likelihood (lnL) at the MLE (see 4.4.3).

Note

`ABLE` expects `tbi` keywords as part of the `ms` command-line **only for** the `task infer` mode. (See also `profile_likelihoods` on pg. 12).

- `bSFS` (T2)
  Accepts a single value specifying the name and location (relative to the config file) of the bSFS output file for the `exact_bSFS` and `conditional_bSFS` cases (see above). The bSFS output is thus for a given demographic model, at a point in parameter space and for a specified number of genealogies.

  ```
  bSFS filename
  ```

  The default name of the output file is `bSFS.txt`.

- `kmax` (T2)
  A single argument following this keyword specifies the maximum number of mutation classes at single nucleotide sites (*i.e.* singletons, doubletons, *etc.*) to be explicitly accounted for in the bSFS. Mutations appearing more frequently in your data than the specified `kmax` are not ignored but rather grouped into a marginal probability class. A maximum of 3 is specified as follows

  ```
  kmax 3
  ```

  Thus, when sampling genealogies/ARGs, `ABLE` will account for 0, 1, 2 and 3 SNPs in all blocks and bundle the probability of observing more the 3 SNPs into a marginal probability.

- `folded` (T1)
  The presence of this keyword instructs `ABLE` to consider the "polarity" *i.e.* account for the ancestral/derived states of alleles with respect to an outgroup and thus use the folded bSFS. If the data is not in a binary format (see Pg. 14) then the first allele at every nucleotide position in the first population is taken to be of the ancestral type.

- `pops` (T3)
  This option takes as a first argument the number of population samples that will be analysed followed by the number of samples per population in the order that they have been specified on the ***ms*** command line

  ```
  pops npops n1 n2 n3 ...
  ```

For a single population example with 5 genomes it should be

```
pops 1 5
```

whereas for a 3 population example with 1, 4 and 2 genomes respectively it should be specified as

```
pops 3 1 4 2
```

It is very easy to account for ghost/unsampled populations in `ABLE` using the `pops` keyword. Lets assume that the third population in the example above was unsampled. In this case, we need to adjust the datafile accordingly (*i.e.* remove the last two sequences from every block) and specify

```
pops 3 1 4 u
```

- `convert_data_to_bSFS` (T2)
  With this option `ABLE` simply converts a `pseudo_MS` format file into the bSFS format. It is advised for users to store data in the bSFS format (especially for large samples) as it is quicker to load and because internally `ABLE` works with the bSFS. When asked to convert data, `ABLE` performs the task, creates a `block_SNPs.txt` file like in the case of `datafile_format pseudo_MS` (see Pg. 6) and terminates. For the conversion you can run `./ABLE config.txt` in the terminal with the contents of `config.txt` as below

```
# (modify the "pops" option according to your sampling)
pops 2 4 4
datafile input_filename
convert_data_to_bSFS output_filename
```

- `start` (T2)/(T3)
  This applies only when `tbi` keywords have been specified as part of the command line (see 4.1) and need to be initialized with numeric values from within the config file.

  - `start all val1 val2 ...`
    When all parameters need to be initialized at once with respect to the "`tbi` order". Let us assume that your demographic model contains three free parameters, `tbi2`, `tbi3` and `tbi7`. If you want to initialize these parameters with `tbi7 = 10`, `tbi2 = 5.2` and `tbi3 = 1`, then

```
start all 5.2 1 10
```

- `start random`
  Initializes all `tbi` keywords with uniformly drawn random values over the respective bounds of each demographic parameter. The default lower and upper bounds for all parameters are 10^{-3} and 5 respectively.
- `start tbi val`
  If a single `tbi` parameter needs to be initialized (*e.g.* `tbi4 = 3`), then

```
start tbi4 3
```

- `global_search` (T2)
  This option sets the global search strategy for the MLE and makes use of the algorithms implemented in the NLopt library.
  - `global_search DIRECT`
    Uses the DIviding RECTangles (DIRECT) algorithm for global optimization.
  - `global_search CRS`
    Uses the Controlled Random Search with local mutation (CRS) algorithm for global optimization.
  - `global_search ISRES`
    Uses the Improved Stochastic Ranking Evolution Strategy (ISRES) algorithm for global optimization.
  - `global_search ESCH`
    Uses the Evolutionary Strategies algorithm by Carlos Henrique da Silva Santos (ESCH) for global optimization.

- `global_search_trees` (T2)
  Accepts a single value which specifies the number of genealogies/ARGs to be sampled during the **global search** of the MLE. The default number of genealogies is 10,000. This corresponds to the precision of the likelihood to be used during this search.

- `local_search_trees` (T2)
  Accepts a single value which specifies the number of genealogies/ARGs to be sampled during the **local search** of the MLE. The default number of genealogies is 15,000. This corresponds to the precision of the likelihood to be used during this search.

- `global_search_evals` (T2)
  Accepts a single value which specifies the number of points in parameter

space that should be explored before concluding the **global search** for the MLE. The default number of evaluations is **2,000** times the number of `tbi` parameters. A warning is issued if the user specifies a number below the default value.

- `local_search_evals` (T2)
  Accepts a single value which specifies the number of points in parameter space that should be explored before concluding the **local search** for the MLE. The default number of evaluations is **1,000** times the number of `tbi` parameters. A warning is issued if the user specifies a number below the default value.

- `global_upper_bound` (T2)
  Accepts a single value which defines the **upper bound** for all `tbi` parameters though this can be undone for certain by specifying individual bounds (see the `bounds` keyword). The default upper bound for all parameters is 5.

- `global_lower_bound` (T2)
  Accepts a single value which defines the **lower bound** for all `tbi` parameters though this can be undone for certain by specifying individual bounds (see the `bounds` keyword). The default lower bound for all parameters is 10^{-3}.

- `skip_global_search` (T1)
  This keyword skips the global search and starts the local search directly with the user-specified start point in parameter space with the help of the `start` keyword (see Pg. 9).

- `skip_local_search` (T1)
  This keyword modifies the default behaviour of `ABLE` and skips the local search after the global search.

- `bounds` (T3)
  Individual parameter bounds during the MLE search can be set with this keyword. The `tbi` parameter needs to be specified as the second keyword and followed by the minimum and maximum bounds respectively. So if you want to impose `0.5 < tbi2 < 4.2`, then

  ```
  bounds tbi2 0.5 4.2
  ```

  The default lower and upper bounds for all parameters are 10^{-3} and 5 respectively.

- `constrain` (T2)
  Presently in `ABLE`, it is possible to specify simple parameter constraints

to enforce biological coherence between two demographic events *e.g.* gene flow is necessarily a more recent event than divergence in a two population model. Multiple constraints can be specified (each on a separate line) and each constraint only accepts two `tbi` parameters. For example, in order to impose the constraint `tbi4 < tbi2`, specify

```
constrain tbi4 tbi2
```

and in this very order.

- `seed_PRNG` (T2)
  Accepts a single value which acts as the starting seed for the Pseudo random Number Generator (PRNG) which follows the Mersenne Twister algortihm (as implemented in the GNU Scientific Library). By default, `ABLE` uses all available threads on a system for computation (see also `set_threads` option), and automatically attributes a different seed to each thread by successively incrementing the user-specified start value by 1. If no value was specified, `ABLE` uses the state of the system clock to generate a seed and then attributes a seed by successive incrementation to each thread.

- `refine_likelihoods` (T2)
  Accepts a single value which specifies the number of genealogies to be sampled for a further refinement of the Monte Carlo *lnL* at the MLE found after a global/local search.

- `profile_likelihoods` (T2)
  Accepts a single value which specifies the number of genealogies to be sampled for calculating the Monte Carlo *lnL* at the user-specified discrete parameter combinations. This is equivalent to specifying `task conditional_bSFS` (see Pg. 7) but differs in that it needs `tbi` keywords and makes use of the `profile` keyword below for specifying the grid.

- `profile` (T3)
  This applies only when the `profile_likelihoods` keyword above has been specified in the config file and `tbi` keywords as part of the command line (see 4.1). It enables the user to explore a grid in n dimensions (i.e. number of `tbi` params) for which `ABLE` outputs a list of likelihoods making it easy to plot the discretized profile likelihood. For each parameter the discrete exploration of likelihoods can be specified as follows `profile tbi val1 val2 ...`

  Below is an example for four `tbi` parameters which tells `ABLE` to explore all combinations starting with `(0.1,0.1,0.6,0.1)` up to `(0.5,0.3,0.7,0.3)`.

```
profile_likelihoods 500

profile tbi2 0.1 0.3 0.5
profile tbi4 0.1 0.2 0.3
profile tbi5 0.6 0.7
profile tbi7 0.1 0.2 0.3
```

Note that unexpected behaviour may result if the `start` keyword (see Pg. 9) is used in conjunction with this option. Additionally, the user can enforce biological constraints on the grid of points using the `constrain` keyword (see Pg. 11).

- `report_likelihoods` (T2)
  Accepts a single value which asks `ABLE` to report the parameter point with the best MLE during the (global or local) search after the specified number of likelihood evaluations.

- `start_likelihood` (T2)
  Accepts a single value which is a user-specified lnL and `ABLE` is asked to check if it can improve over this value during a local search. This option applies only when the `skip_global_search` keyword has been specified.

- `no_bSFS_file` (T1)
  This asks `ABLE` not to output a bSFS file. Only standard information such as the lnL and computation time are written to the console.

- `print_correction_factor` (T1)
  This option asks `ABLE` to print the correction factor (≥ 1) which penalizes the likelihood at a point in parameter space. These situations can arise when the number of genealogies specified by the user are insufficient for explaining all the bSFS configurations present in the data. Or when you are attempting to fit the data without recombination when the data bSFS contains configurations which violate the four gamete test. The printing is only activate when `task conditional_bSFS` (see Pg. 7) and a value greater than 1 indicates that the penalization is in effect.

- `set_ftol_abs` (T2)
  Accepts a single value which sets the tolerance in terms of the difference in absolute value between successive evaluations at points in parameter space during the local search. Let ϵ be the user provided value and assume that the local search has evaluated the likelihood function, $f(x)$, at points x_1 and then x_2. If the condition $|f(x_2) - f(x_2)| < \epsilon$ is satisfied, the search is terminated and the MLE is reported by `ABLE`.

- `set_threads` (T2)
  Accepts a single value indicating the number of threads that `ABLE` needs to spawn for a parallel computation of the bSFS likelihoods.

4.3 Data format

`ABLE` accepts data in simple text files which can be in either of two formats : `pseudo_MS` or `bSFS` (see Pg. 6). If you have your data in other formats (*e.g.* VCF,...), then converting it into the `pseudo_MS` should be very easy. The order in which population samples need to be input closely resembles the ***ms*** format. Briefly, the samples from each population for which you will have attributed an *a priori* order (`pops` keyword, see Pg. 8) should be listed.

By default, the `pseudo_MS` format expects sequence blocks using only the genotype information (*i.e.* `A/T/G/C/N`). If you have prior information regarding the ancestral/derived states at each segregating site you can specify the blocks in a binary format (using the `allele_type` option, see Pg. 7), in which case all sequences must be in this format. Nucleotide positions containing a `N` (*e.g.* due to missing information) will be completely ignored as are tri/quadri-allelic SNPs.

4.3.1 The `pseudo_MS` genotype format

Each block begins with a `//` followed by an optional header which can span any number of lines. Below is an example of three sequence blocks – a Chr7 block, a fictional monomorphic block and from Chr4 respectively – from the accompanying 2kb Orangutan dataset (`/data` folder). The first four samples (2 diploids) are from the Bornean population and the rest are from Sumatran individuals. The corresponding population order is `pops 2 4 4` (see Pg. 8).

Note

Headers **may not begin** with either of `A`, `T`, `G`, `C` or `N` characters! Use a `#` to start a line if necessary.

```
//
7_41030400-41032000
TCATCTG
TCATCTG
GCATTGT
GCATTGT
GCAGCGG
TCTTCTG
GTATCGG
```

```
GTATCGG

//
7_xxxxxxxx-xxxxxxxx (monomorphic_block_example)

//
4_179188400-179190000
ATGGAGT
ATGGAGT
ACAGAGC
GTGGGGC
ATGGAAT
GTGGGGT
ATGAGGT
ATGAGGT
```

4.3.2 The `pseudo_MS` binary format

Now assume that the ancestral states in the previous example (4.3.1) were given by the genotypes of the first sequence of each block. The binary equivalent is obtained by replacing the ancestral allele with `0` and derived allele with `1` and should look like the following.

Note

Headers **may not begin** with either of `0`, `1` or `N` characters! Use a `#` to start a line if necessary.

```
//
7_41030400-41032000
0000000
0000000
1000111
1000111
1001010
0010000
1100010
1100010

//
7_xxxxxxxx-xxxxxxxx (monomorphic_block_example)

//
```

```
4_179188400-179190000
0000000
0000000
0110001
1000101
0000010
1000100
0001100
0001100
```

4.4 Examples

In this section we shall refer to three demographic models (Fig. 1), also considered in the accompanying bioRxiv draft (although some details may vary) for illustrating the different tasks performed of `ABLE` (see Pg. 7). Note however that the following assumes a good working knowledge of the ***ms*** command line options.

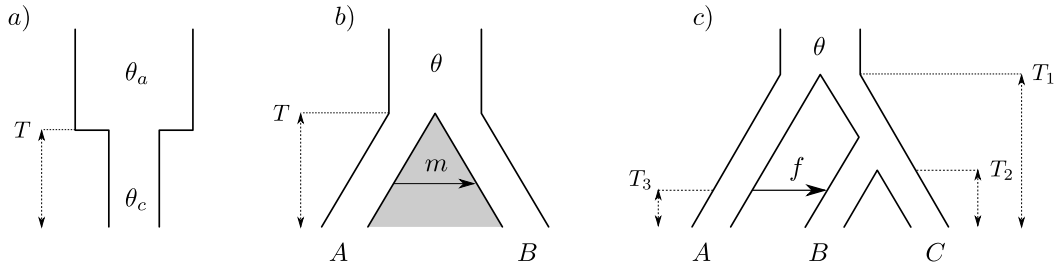

Figure 1: Demographic models previously considered in the accompanying bioRxiv draft. (*a*) a single population with a sudden reduction in N_e, (*b*) isolation between populations A and B followed by continuous unidirectional migration (from A to B) at rate M migrants per generation and (*c*) isolation between three populations A, B and C followed by unidirectional admixture of a fraction f from A to B.

4.4.1 The *exact* bSFS

Let us consider a single population sample of size 4 which doubled its effective population size at scaled time $T = 0.2$ in the past with $\theta_c = 1$ and $\theta_a = 2$ (*see* Fig. 1a). We would like to calculate the exact *folded* bSFS (see Pg. 7) using 100K genealogies. The corresponding command line would look like

```
./ABLE 4 100000 -t 1 -eN 0.2 2 -T config_1pop_exact.txt
```

along with the contents of `config_1pop_exact.txt` as below

```
pops 1 4

kmax 4
folded
task exact_bSFS
bSFS exact_bSFS.txt

seed_PRNG 98368183
```

The `kmax 4` (see Pg. 8) is not a necessary option here and only serves to restrict the size of the bSFS by accounting explicitly for up to 3 SNPs per branch class in every block. A marginal probability class is calculated for all bSFS configurations having more then 3 SNPs per branch class.

4.4.2 The *conditional* bSFS

This example calculates the probabilities of observing only the bSFS configurations present in the data for a three population model. The parameter values used here are $f_{A\to B} = 0.04$, scaled population size $\theta = 2.432$, scaled times $T_3 = 0.0625$, $T_2 = 0.075$ and $T_1 = 0.3$ (*see* Fig. 1c).

```
./ABLE 3 1000000 -t 2.432 -I 3 1 1 1 -es 0.0625 1 0.96 -ej
   0.0625 4 3 -ej 0.075 1 2 -ej 0.3 2 3 -T
   config_3pop_conditional.txt
```

And `config_3pop_conditional.txt` contains :

```
pops 3 1 1 1

task conditional_bSFS
datafile data_filename_to_be_specified_here.txt
bSFS conditional_bSFS.txt

seed_PRNG 12468192
```

4.4.3 Inference with the bSFS

We now consider a two population example with 3 genomes in population A and 2 genomes in B (*see* Fig. 1b) and blocks of size 500bp. This is a four parameter model with unidirectional gene flow at rate m after a split at time T, with all scaled populations size parameters equal to θ and scaled intra-block recombination rate

ρ. The parameters *"to be inferred"* : θ, ρ, m and T have been respectively specified as `tbi1`, `tbi2`, `tbi3` and `tbi4` in the command line below. Note that the position corresponding to the number of genealogies to be sampled during inference (`xxx` below) is entirely ignored by `ABLE` and should only be specified in the config file.

```
./ABLE 5 xxx -t tbi1 -r tbi2 501 -I 2 3 2 -m 1 2 tbi3 -ej tbi4
   1 2 -T config_2pop_infer.txt
```

The contents of `config_2pop_infer.txt` below starts a global search immediately followed by a local search with the resulting MLE form the former.

```
# general options
# -----------------------
pops 2 3 2
task infer
datafile data_filename_to_be_specified_here.txt
seed_PRNG 29468147

# global search options
# -----------------------
global_search CRS
global_search_trees 50000
global_search_evals 7000
report_likelihoods 1000

# parameter constraints
# -----------------------
global_upper_bound 15
global_lower_bound 1e-2
bounds tbi1 1e-2 1
bounds tbi2 1e-2 1

# local search options
# -----------------------
local_search_trees 50000
refine_likelihoods 1000000
```

Index of keywords