`msbgs` - a program for generating samples under background selection models

Kai Zeng

Department of Animal and Plant Sciences, University of Sheffield, UK
k.zeng@sheffield.ac.uk

Version 1.01

1 Introduction

`msbgs` is a simulation program for generating variability under background selection models using the coalescent framework first described in Zeng and Charlesworth (2011). It can accommodate biologically important factors including recombination (crossover), variation in selection coefficient against deleterious mutations across sites, and changes in population size (Zeng, 2013); more are being incorporated (e.g., population structure and migration; Zeng and Corcoran, 2015). The program is an extension of Hudson's `ms` (Hudson, 2002); whenever possible, the command line switches follow the conventions used in `ms`. As in `ms`, an infinite-sites model of mutation is assumed.

2 Citations

Zeng, K. (2013). A coalescent model of background selection with recombination, demography and variation in selection coefficients. *Heredity (Edinb)*, 110(4):363–71.

Zeng, K. and Charlesworth, B. (2011). The joint effects of background selection and genetic recombination on local gene genealogies. *Genetics*, 189(1):251–66.

Zeng, K. and Corcoran, P. (2015). Modelling the effects of selection against recurrent deleterious mutations on differentiation patterns in subdivided populations. *Submitted*.

3 Download and compilation

The program is written in C and can be run on Unix/Linux/Mac. It should be possible to use it on Windows if Cygwin is installed, but this has not been tested. All the files are contained in `msbgs.tar.gz`. To decompress, issue

```
tar -xf msbgs.tar.gz
```

which will generate a folder named `msbgs`. There are two ways to compile the program, depending on which C compiler is used. Before compiling, it is necessary to install the GNU Scientific Library (GSL):

```
http://www.gnu.org/software/gsl
```

3.1 Installing GSL

`msbgs` was developed using GSL v1.15 (using newer versions should not be a problem, but this has not been tested). Compiling and installing GSL are relatively straightforward. To unpack, issue

```
tar -xf gsl-1.15.tar.gz
```

Enter the folder generated, and issue

```
./configure --disable-shared --prefix=$DIR
```

where `$DIR` should be replaced by the full path of the folder in which the final library files are to be stored (e.g., `--prefix=/home/gsl-1.15`). When the above is done, issue

```
make
```

This will take a little while. When done, issue

```
make check > test.log 2>&1
```

This will carry out tests and save the results to `test.log`. When the process is completed, open `test.log`, and go over it carefully. There should be no errors or failures. Finally, issue

```
make install
```

This will copy the library files to the folder specified by `$DIR`. Within this folder, there should be a folder named `lib`, which contains a file named `libgsl.a`.

3.2 Compiling using GCC

GCC is freely-available and should exist on most computers. Issue

```
gcc -v
```

in the command line to check availability. To compile, enter the `msbgs` folder and open the file named `compile_gcc.sh` in a text editor. The first two lines starts with `GSL_INCLUDE=` and `GSL_LIB=`, respectively. Go to the GSL library folder (i.e., the one specified by `$DIR`). Enter the folder named `include` and issue `pwd`. Copy the path printed on screen and insert it after `GSL_INCLUDE=` (Note: It is important that there is no space on either side of `=`). In the GSL library folder there is another folder named `lib`. Use the same method to find out its path and insert it after `GSL_LIB=`. To compile, issue

```
sh compile_gcc.sh
```

An executable named `msbgs` will be generated. A binary compiled on an Intel 64-bit Linux platform is included in the package (`msbgs_gcc_x86_64_Linux`).

3.3 Compiling using the Intel C compiler

Intel's proprietary C compiler, ICC, is known to generate more efficient code than GCC. A performance comparison indicates that the program compiled by ICC is significantly more efficient than that compiled by GCC (~20%). Issue

```
icc -v
```

to check availability. Open `compile_icc.sh` and update `GSL_INCLUDE=` and `GSL_LIB=` as above. To compile, issue

```
sh compile_icc.sh
```

An executable named `msbgs_icc` will be generated.

4 The basic command line for generating *neutral* variants

`msbgs nsam leng nrep -t` θ `-r` ρ `-sel sel_def -region region_def [-seed seed]`

These arguments and switches should appear in *exactly* the same order as displayed above.

`nsam` (≥ 2) is the sample size. `leng` (≥ 1) is the size (in bp) of the entire simulated region (including both selected and neutral sites; see below). `nrep` (≥ 0) is the number of simulations to carry out.

$\theta = N_r u$, where N_r is the reference *haploid* population size and u is the mutation rate per site per generation (i.e., $N_r = 2N$, where N is the population size of a *diploid* organism). Note that time is scaled in units of N_r generations; so as all other scaled parameters (see below).

$\rho = N_r r$, where r is the recombination rate per site per generation. Note that `-r` should still be included even when the simulated region is completely link:

```
-r 0
```

This is different from `ms`.

`sel_def` is a quotation-mark-delineated string, giving a list of scaled selection coefficients. When there is more than one scaled selection coefficient, they should be separated by commas. Here are some examples:

```
-sel "10"
```

Notes: a single selection coefficient; the quotation marks can be omitted in this case.

```
-sel "10,20,30"
```

Notes: three selection coefficients, denoted by γ_0, γ_1, and γ_2, respectively (see below); the quotation marks are compulsory.

```
-sel "0"
```

Notes: zero is allowed only if *exactly* one element is in the list. In this case, all sites in the simulated region are neutral, and `msbgs` is equivalent to `ms`. It is worth pointing out that `msbgs` seems to be more efficient than `ms` in carrying out equivalent neutral simulations (up to an order of magnitude in the cases tested), thanks to improvements in memory management.

The i-th element of the list is denoted by γ_i, which is defined as $N_r s_i$, where s_i is the *haploid* selection coefficient. s_i is equivalent to the selection coefficient in the following *diploid* model: let the fitnesses of the three genotypes A_0A_0, A_0A_1, and A_1A_1 be 1, $1-s_d$, and $1-2s_d$, respectively; then $s_i = s_d$.

`region_def` is a quotation-mark-delineated string defining the sites under selection and their associated selection coefficients; sites not included in `region_def` are neutral. To conduct neutral simulations, we should set

```
-sel "0" -region "neutral"
```

Notes: all the quotation marks can be omitted in this example.

Otherwise, `region_def` should contain the same number of elements as in `sel_def`. Consider the following example:

```
-sel "10,20" -region "[element_0];[element_1]"
```

`element_0` specifies the positions of the sites that are under purifying selection with selection coefficient $\gamma_0 = 10$; the same applies to `element_1`. Note that the first site in the simulated region has index 0, and the last site has index `leng - 1`. Each element is composed of a list of comma-separated sub-elements. There are two types of sub-elements, `x` and `(start_step_cn)`. `x` represents an integer. `(start_step_cn)` expands to `start` $+ j \times$ `step`, where `start` (≥ 0) defines the starting position, `step` (≥ 1) is the step size, and j ranges from 0 (inclusive) to `cn` (≥ 1; inclusive). These two types of sub-elements can appear multiple times in arbitrary order. For example, the following examples are equivalent, all setting the first five sites in the simulated region to have $\gamma = 10$ and the next five sites to have $\gamma = 20$:

```
-sel "10,20" -region "[0,1,2,3,4];[9,8,7,6,5]"
-sel "20,10" -region "[(5_1_2),(8_1_1)];[0,(1_1_2),4]"
```

The following is an example where there is only one element in `sel_def`:

```
-sel "10" -region "[0,1,2,3,4]"
```

Notes: the first pair of quotation marks can be removed, but the second pair is mandatory.

It is, however, important to note that each site can only be assigned once; otherwise an error will be thrown and the program will be terminated. Moreover, at least one site should be set for each γ defined by `-sel sel_def`.

The switch `-seed seed` is optional. If it is not included, then the random number generator will be seeded by using a number originated from the current time of the system. The seed used will be printed in the output. If the same seed and other parameters are used, the simulation results will always be identical.

In sum, the basic command line specifies simulations based on a model with a randomly mating population of constant size N_r. Although it is possible to simultaneously generate variants at both selected and neutral sites (Zeng, 2013), this function has not been implemented in this release.

5 The output

The simulated data are printed to the screen. The output format is similar to that of `ms`. Assume that we run an experiment using the following command:

```
msbgs 4 1200 2 -t 0.005 -r 0 -sel 10 -region "[(0_3_399),(1_3_399)]" -seed 17
```

which simulates a 1.2 kb genomic region composed of triplets within each of which the first two sites are selected and the third is neutral. The following will be printer to screen:

```
msbgs 4 1200 2 -t 0.005 -r 0 -sel 10 -region [(0_3_399),(1_3_399)] -seed 17

seed:  17

//
mean_coal_rate:  1.491824698
segsites:  4
positions:  0.062142014 0.864195261 0.876846261 0.947285472
1000
0110
0110
0001

//
mean_coal_rate:  1.491824698
segsites:  5
positions:  0.057098017 0.144413964 0.152220945 0.419503833 0.719798666
00001
00001
11110
00001
```

Note that the quotation marks required by the various command line arguments are omitted in the output.

The first line of the output is the command line. `seed:` shows the random seed. Each sample is preceded by a line with just `//` on it. `mean_coal_rate` gives the mean pairwise coalescent rate calculated using all the coalescent event that have occurred during the course of the simulation. (Note: The fact that both replicates in the above example have the same `mean_coal_rate` is a coincidence.) Let μ denote the mean of `mean_coal_rate` across many replicates; it is defined with respect to N_r. Under neutrality, the pairwise coalescent rate of a population of size N_r is $\mu = 1$, whereas a population of size $N_r/2$ has rate 2 when time is measured in units of N_r generations. N_r/μ is the average size (in terms of the number of *haploid* individuals) of a genetic background within which a coalescent event occurs (Eq. (11) of Zeng and Charlesworth, 2011; and supplementary Eq. (S2) of Zeng, 2013). N_r/μ tends to be small when selection is very weak and/or there are a large number of selected sites, in which case the coalescent model tends to be inaccurate. Note that μ is analogous to supplementary Eq. (S2) of Zeng (2013), but is easier to obtain, and yet seems to be effective (based on a less extensive test than the one used by Zeng (2013)). Simulations suggest that the algorithm is reliable when N_r/μ is not small (>10, say) and $N_e s$ is > 1 (see **Obtaining $N_e s$** below) for panmictic populations (Zeng, 2013). For subdivided populations, $N_e s$ larger than about 5 is often necessary (Zeng and Corcoran, 2015). Supplementary Eq. (S2) of Zeng (2013) can be

calculated by making use of the fact that expected the number of neutral segregating sites, $E(S)$, equals to θLT, where θ is defined above, L is the total number of neutral sites in the region, and T is the total branch length (in units of N_r generations). `segsites` gives the number of segregating sites in this sample. `positions` contains a list of real numbers that tell the positions of the segregating sites. These positions are defined with respect to the entire simulated region, which is now represented by the interval [0, 1], such that the first site (with index 0) is represented by [0, 1/`leng`], where `leng` is the size of the region. These positions are generated randomly. When only neutral variants are simulated numbers in positions will only fall within the neutral regions. Each of the `nsam` lines that follow represents data from a haploid individual in the sample, with 0 and 1 representing the ancestral and derived allele, respectively.

6 Obtaining $N_e s$

$N_e s$ is a measure of the efficacy of purifying selection. In models without recombination, it is known that $N_e s \gg 1$ can prevent the irreversible accumulation of deleterious mutations due to the process of Muller's ratchet (Jain, 2008; Neher and Shraiman, 2012). `msbgs` can calculate $N_e s$ for each selected site and report the smallest value. Specifically, for each selected site, the value is equal to $BN_r s = B\gamma$, where B is the reduction due to background selection (Hudson and Kaplan, 1995; Nordborg et al., 1996). Note that B is equivalent to letting time go to infinity in the time-dependent solutions obtained by Nicolaisen and Desai (2012, 2013). If the population size is xN_r due to past demographic events (see below), then the returned values can be multiplied by x to obtain corresponding values for that period. If the period of population size reduction (such that $N_e s$ for that period becomes small) is very brief, there is some evidence that the effect on the reliability of the coalescent model is not significant. Nonetheless, caution should be exercised whenever $N_e s$ is small.

To obtain $N_e s$, append the switch `-nes` after those listed in *The basic command line*. But `-nes` and the switches listed below in **Past demographic events** can appear in any order. There are three options:

`-nes min`
Print the minimum $N_e s$ value.

`-nes selected`
Print $N_e s$ values for all selected sites across the simulated region.

`-nes B` Print B values for all sites (neutral and selected) across the simulated region. B, defined as N_e/N_r, is the reduction caused by background selection (see Eq. (5) of Hudson and Kaplan, 1995).

If these switches appear more than once, the last one overrides the previous ones. The results of the calculation will be printed to screen and will appear after `seed:` but before the first simulated sample (if any).

```
msbgs 4 1000 0 -t 0.01 -r 0.01 -sel 10 -region "[(0_2_499)]" -nes min
```

Notes: `nrep` is set to 0, so no simulation is carried out. `nsam` has no effect on the calculation of $N_e s$.

The above command produces the following output:

```
msbgs 4 1000 0 -t 0.01 -r 0.01 -sel 10 -region [(0_2_499)] -nes min

seed:  1434296087

nes_min:  7.165309744
```

7 Past demographic events

These switches must appear after those listed in **The basic command line**. The events must be given in ascending order with respect to the time of occurrence. If multiple events occur at the same time, then they are processed according to the order they appear in the command line. Time is measured in units of N_r generations.

`-eN "t,x"`
Set all subpopulations size to $x \times N_r$ at time `t`. It is important that the quotation marks are included. Consider the following population bottleneck model:

```
msbgs 2 1200 2 -t 0.005 -r 0 -sel 50 -region "[(0_3_399),(1_3_399)]" -eN "0.1,
0.1" -eN "0.15, 1"
```

It starts with a population of size N_r. At time 0.1 before the present, the population size reduces to $0.1N_r$, and goes back to N_r from time 0.15 onwards.

8 Other options

The following are some additional options. They must be added after those listed in *The basic command line*.

`-ms_format`
This makes the output format identical to `ms`. See the following example:

```
msbgs 4 1200 1 -t 0.005 -r 0 -sel 10 -region [(0_3_399),(1_3_399)] -ms_format
1436344125 1436344125 1436344125

//
segsites:  11
positions:  0.154226921 0.329982924 0.394247544 0.402126662 0.437045401
0.494249863 0.589434382 0.654249850 0.679389570 0.957324390 0.981938652
10000001100
10000001101
00100101110
01011010000
```

9 Acknowledgements

I thank Pádraic Corcoran and Toni Gossmann for their assistance in the testing of this program.

10 References

Hudson, R. R. (2002). Generating samples under a wright-fisher neutral model of genetic variation. *Bioinformatics*, 18(2):337–8.

Hudson, R. R. and Kaplan, N. L. (1995). Deleterious background selection with recombination. *Genetics*, 141(4):1605–17.

Jain, K. (2008). Loss of least-loaded class in asexual populations due to drift and epistasis. *Genetics*, 179(4):2125–34.

Neher, R. A. and Shraiman, B. I. (2012). Fluctuations of fitness distributions and the rate of muller's ratchet. *Genetics*, 191(4):1283–93.

Nicolaisen, L. E. and Desai, M. M. (2012). Distortions in genealogies due to purifying selection. *Mol Biol Evol*, 29(11):3589–600.

Nicolaisen, L. E. and Desai, M. M. (2013). Distortions in genealogies due to purifying selection and recombination. *Genetics*, 195(1):221–30.

Nordborg, M., Charlesworth, B., and Charlesworth, D. (1996). The effect of recombination on background selection. *Genet Res*, 67(2):159–74.

11 Change log

Version 1.01
Added `-ms_format`.

Version 1.00
Initial release.