GADMA user manual

Corresponding to version 1.0.0

Ekaterina Noskova

Contents

1 Introduction

Welcome to GADMA!

GADMA implements methods for automatic inferring joint demographic history of multiple populations from the genetic data.

GADMA is based on two open source packages: the ∂a∂i developed by Ryan Gutenkunst [`https://bitbucket.org/gutenkunstlab/dadi/`] and the *moments* developed by Simon Gravel [`https://bitbucket.org/simongravel/moments/`].

In contrast, GADMA is a command-line tool. It presents a series of launches of the genetic algorithm and infer demographic history from Allele Frequency Spectrum of multiple populations (up to three).

2 Getting help

Please don't be afraid to contact me for different problems and offers via email ekaterina.e.noskova@gmail.com. I will be glad to answer all questions.

3 Hands on

Test case
GADMA has test case for simple demographic model for 1 population: just constant size of 10000 individuals in population. To run test case type:

```
$ gadma --test
```

Example 2
Suppose we have SNP data for two populations. Data is in ∂a∂i's SNP format in file `snp_data.txt`. Suppose we want to get all output in some `gadma_output` directory:

```
$ gadma -i snp_data.txt -o gadma_output
```

Example 3
We didn't specify AFS size or labels for populations, they are taken automaticaly from the input file.We can see parameters file of our run in `gadma_output/params` file. We see that there are

```
# gadma_output/params
...
Population labels : pop_name_1 , pop_name_2
Projections: 18, 20
...
```

But we know that spectrum should be 20×20! To specify size of AFS we need to create parameters file and set `Projections`:

```
# params_file
Projections : 20,20
```

Also we can change order of populations. We should add:

```
# params_file
Projections : 20,20
Population labels : pop_name_2, pop_name_1
```

If we want to rename populations, we should change names in `snp_data.txt` file.

Now assume we want to get the simplest demographic model that we can as faster as we can. We will tell GADMA that we don't need no other dynamics of population sizes except sudden (constant) population size change and that we want to use *moments* library.

We add corresponding string to parameters file and now it looks like:

```
# params_file
Projections : 20,20
Population labels : pop_name_2, pop_name_1
Only sudden : True
Use moments or dadi : moments
```

To run GADMA we need to specify `-p/--params` command-line option in cmd:

```
$ gadma -i snp_data.txt -o gadma_output -p params_file
```

Example 4

Consider some AFS file `fs_data.fs`. There is spectrum for three populations: YRI, CEU, CHB. However axes are mixed up: CHB, YRI, CEU. To run GADMA we should order them from most ancient to last formed:

```
# params_file
Population labels : YRI, CEU, CHB
```

We want to allow exponential growth (it is default behaviour) and have some extra change in size of the ancient population. To do so we should specify `Initial structure`. It is list of three numbers: first — number of time intervals before first split (we want here 2); second — number of time periods betseen forst and second split events (at least 1); third — number of time periods after second split.

```
# params_file
Population labels : YRI, CEU, CHB
Initial structure : 2,1,1
```

Also we can put information about input file and output directory to our parameters file:

```
# params_file
Input file : fs_data.fs
Output directory : gadma_output
Population labels : YRI, CEU, CHB
Initial structure : 2,1,1
```

Now we can run GADMA the following way:

```
$ gadma -p params
```

Example 5
We have our GADMA launch interrupted for some reason. We want to resume it:

```
$ gadma --resume gadma_output
```

where `gadma_output` is output directory of previous run. We can find resumed run in `gadma_output_resumed`

Example 6
Our launch was finished, we used ∂a∂i with default grid size which GADMA determines automatically if it is not specify by user. We found out that it would be better to find some models using greater number of grid points in ∂a∂i scheme, but we want to take final models from previous run:

```
# params_file
Pts : 40, 50, 60 #Greater value of grid size than it was
```

And run GADMA:

```
$ gadma --resume gadma_output --only_models -p params
```

`--only_models` tell GADMA to take from `gadma_output` final models only.

There is another way to do the same:

```
# params_file
Resume from : gadma_output
Only models : True
Pts : 40, 50, 60 #Greater value of grid size than it was
```

And run GADMA the following way:

```
$ gadma -p params
```

Example YRI, CEU
GADMA has example of full parameters file `example_params`, which is preseted here too at the end of the manual. To run GADMA with this parameters one should just run from the GADMA's home directory:

```
$ gadma -p example_params
```

4 Input data

GADMA handles two formats of input files:

- Frequency spectrum file format (.fs or .sfs),

- SNP data format (.txt).

These data formats should be familiar to one who was used ∂a∂i or *moments*.
Input file can be specified to GADMA with two ways:

1. Through command-line option `-i/--input`:

```
$ gadma -i fs_file.fs -o out_dir
```

or

```
$ gadma --input snp_file.txt -o out_dir
```

2. In parameters file `params_file`:

```
# params_file

# Input file path
Input file : fs_file.fs
...
```

Extra information about input AFS can also be put in parameters file. For example, AFS can be projected to smaller size with `Projections` option, populations can be named or their order can be changed with `Population labels` option. If parameters file does not contain some options, they are automaticaly pulled out from the input file.

```
# params_file

# Input file path
Input file : fs_file.fs

# (New) size of the AFS
Projections : 20,20

# Labels of populations
Population labels : CEU, YRI
...
```

GADMA can be launched with parameters file the following way:

```
$ gadma -p params_file -o out_dir
```

4.1 VCF data format

If one have .vcf file, it can be converted it into .sfs file using `https://github.com/isaacovercast/easySFS`.

4.2 Frequency spectrum file format

Each file begins with any number of comment lines beginning with `#`. The first non-comment line contains P integers giving the dimensions of the FS array, where P is the number of populations represented. For a FS representing data from 4x4x2 samples, this would be `5 5 3`. (Each dimension is one larger than the number of samples, because the number of observations can range, for example, from 0 to 4 if there are 4 samples, for a total of 5 possibilities.) On the same line, the string `folded` or `unfolded` denoting whether or not the stored FS is folded.

The actual data is stored in a single line listing all the FS elements separated by spaces, in the order fs[0,0,0] fs[0,0,1] fs[0,0,2]...fs[0,1,0] fs[0,1,1].... This is followed by a single line giving the elements of the mask in the same order as the data, with `1` indicating masked and `0` indicating unmasked.

4.3 SNP data format

```
Human Chimp Allele1 YRI  CEU  Allele2 YRI CEU Gene  Position
ACG   ATG   C       29   24   T       1   0   abcb1 289
CCT   CCT   C       29   23   G       3   2   abcb1 345
```

Listing 1: Example of SNP file format

The data file begins with any number of comment lines that being with `#`. The first parsed line is a column header line. Whitespace is used to separate entries within the table, so no spaces are allowed within any entry. Individual rows make be commented out using `#`.

The first column contains the in-group reference sequence at that SNP, including the flanking bases. If the flanking bases are unknown, they can be denoted by `-`. The header label is arbitrary.

The second column contains the aligned outgroup reference sequence at that SNP, including the flanking bases. Unknown entries can be denoted by `-`. The header label is arbitrary.

The third column gives the first segregating allele. The column header must be exactly `Allele1`.

Then follows an arbitrary number of columns, one for each population, each giving the number of times Allele1 was observed in that population. The header for each column should be the population identifier.

The next column gives the second segregating allele. The column header must be exactly `Allele2`.

Then follows one column for each population, each giving the number of times Allele2 was observed in that population. The header for each column should be the population identifier, and the columns should be in the same order as for the Allele1 entries.

Then follows an arbitrary number of columns which will be concatenated with `_` to assign a label for each SNP.

The `Allele1` and `Allele2` headers must be exactly those values because the number of columns between those two is used to infer the number of populations in the file.

5 Specifying scheme

GADMA use either ∂a∂i or *moments* to simulate expected AFS from the demographic model. By default ∂a∂i is used. To use ∂a∂i it is recommended to check value of the `Pts` option in parameters file. `Pts` is sequence of three numbers, each of which is equal to the number of points in grid size. The greater numbers are, the more accurate ∂a∂i numerical solution of partial differential equation is, however, it would take more time. By default, GADMA takes `Pts : n, n + 10, n + 20`, where `n` — is the maximum sample size among populations of interest.

moments library does not need `Pts` to be specified. To change ∂a∂i to *moments* scheme, specify option in parameters file:

```
# params file
...
Use moments or dadi : moments
...
```

6 Specifying a model

6.1 Dynamics of population size change

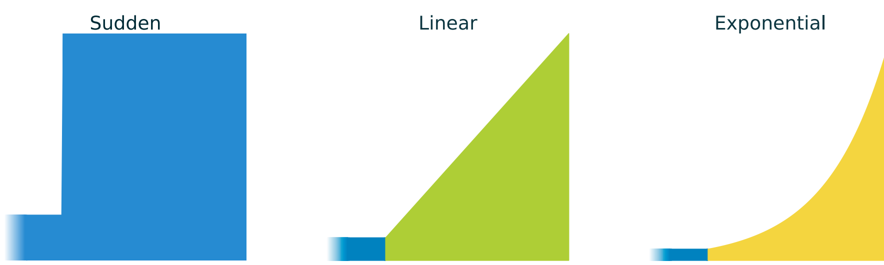

Figure 1: Three main demographic dynamics of population size change.

In GADMA size of population can be changed due to one of three dynamics: sudden change, linear change and exponential change of the effective population size (Figure 1).

In order to infer demographic model with sudden changes of populations sizes only, option `Only sudden` in parameters file should be set to `True`:

```
# params_file
...
Only sudden : True
...
```

By default, this option is set to `False` and dynamics are found like other parameters of the demographic model.

6.2 Specifying structure of the model

GADMA infers a demographic model from an AFS with nothing required from the user, except simple information, that determines how much the detailed model is required — the **structure of the model**.

Assume a division of one population into two new subpopulations and fixed temporal order of the populations: from the ancient to the most recently formed population.

We can divide time of our model into splits events and time intervals, during which a certain dynamics of change of effective size is maintained for each population and migration rates are constant. The number of split events is one less than the number of populations under consideration. Now we can define the concept of the structure of the demographic model:

Structure of the demographic model is:

- the number of time intervals in case of one population;
- the number of time intervals as those that occur before and after a single split, in the case of two populations;
- the number of time intervals prior to the first split, those between the first and second split, and the ones after the second split, in the case of three populations.

For example, we can divide time of the model on the Figure 2 to 4 time intervals: T_1, T_2, T_3 and T_4, and two populations' splits: S_1 and S_2. The structure of this model is $2, 1, 1$, because two intervals (T_1 as T_2) before first split S_1, one interval (T_3) between first and second splits and one interval (T_4) after second split S_2.

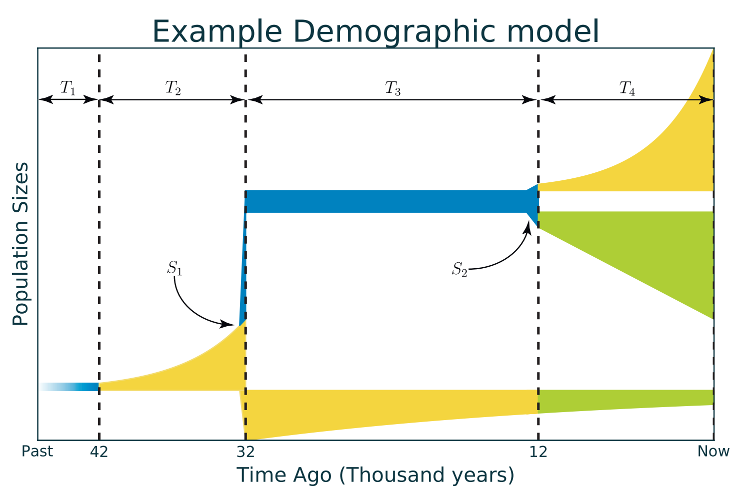

Figure 2: Example of representation of demographic model. Time is on the axis of abscissa and population size is on the axis of ordinates. The structure of that model is (2,1,1). Here colors are just a formality and are used for highlighting different demographic dynamics.

6.2.1 Initial structure

To specify Structure of the inferred model one should set `Initial structure` in the parameters file:

```
# params file
...
Initial structure : 2
...
```

or

```
# params file
...
Initial structure : 2,1
...
```

or

```
# params file
...
Initial structure : 2,1,1
...
```

By default the simplest structure is used (1 or 1,1 or 1,1,1).

6.2.2 Final structure

It is also possible to start with more simple structure in order to get more complex. To do so one should specify option `Final structure` in the parameters file. For example:

```
# params file
Input file : some_2d_fs.fs

Initial structure : 1,1
Final   structure : 2,1
...
```

During this parameters file GADMA will find parameters for demographic model with 1,1 structure, then increase structure for 2,1 and find parameters for the model with this structure. Found parameters for more simple structure (in this case it is 1,1) are used in search of the parameters of more complex structure (2,1).

Note: Structure increases random, so if one specify initial structure to 1,1 and final to 2,2, it is not garanteed to finad optimal parameters for demographic models with structures between 1,1 and 2,2, i.e. intermediate state can be either 1,2 or 2,1.

Recommendation: Use scheme with increase of the structure, as it produce more stable solutions.

Recommendation: Choose not very large values for model's structure. Final structure is recommended to be differ by one element from initial structure, for example, 1,1 and 2,1; 1,2,1 and 2,2,1.

6.3 Specifying model in details

It is also possible to use Genetic algorithm from GADMA to a proposed model that is defined as Python function using ∂a∂i or *moments*. It is the way that ∂a∂i and *moments* work with demographic models inference. To understand how to specify model like that one can read manuals to the packages.

For example, consider simple bottleneck model for one population: at time `TF + TB` in the past, an equilibrium population goes through a bottleneck of depth `nuB`, recovering to relative size `nuF`:

```
def bottleneck(params, ns, pts):
    nuB, nuF, TB, TF = params
    xx = dadi.Numerics.default_grid(pts)

    phi = dadi.PhiManip.phi_1D(xx)
    phi = dadi.Integration.one_pop(phi, xx, TB, nuB)
    phi = Integration.one_pop(phi, xx, TF, nuF)

    fs = dadi.Spectrum.from_phi(phi, ns, (xx,))
    return fs
```

To run optimization from GADMA one need to run optimization function just like in ∂a∂i and *moments*:

```
# Import GADMA's optimization function:
import gadma

# Specify input data and its parameters:
data = dadi.Spectrum.from_file("fs_filename.fs")
ns = data.sample_sizes # size of AFS
pts = [40,50,60] # grid size for dadi

# Wrap our bottleneck function:
func_ex = dadi.Numerics.make_extrap_log_func(bottleneck)

# Specify upper and lower bounds for parameters:
upper_bound = [100, 100, 3, 3]
lower_bound = [1e-2, 1e-2, 0, 0]

# Run optimizations:
# Beginning GADMA optimization
popt = gadma.Inference.optimize_ga(4, data, func_ex, pts_l,
                                   lower_bound=lower_bound,
                                   upper_bound=upper_bound)
# Beginning local optimization from dadi
popt = dadi.Inference.optimize_log(popt, data, func_ex, pts_l,
                                   lower_bound=lower_bound,
```

```
                                        upper_bound=upper_bound,
                                        verbose=len(p0), maxiter=3)

print('Found parameters: {0}'.format(popt))
```

Note: No initial parameters `p0` are set in `gadma.Inference.optimize_ga` function as GADMA optimization is global search algorithm. However it is possible to specify `p0`:

```
# Initial parameters can be set too:
p0 = [0.01, 1.5, 0.2, 0.2]

# Beginning GADMA optimization
popt = gadma.Inference.optimize_ga(4, data, func_ex, pts_l, p0=p0
                                        lower_bound=lower_bound,
                                        upper_bound=upper_bound)
```

If one want to find other parameters fo `gadma.Inference.optimize_ga` function:

```
>>> import gadma
>>> help(gadma.Inference.optimize_ga)
```

7 Several repeats and parallel computing

By default GADMA run optimization that uses Genetic algorithm once. However it is recomended to run optimization several times and get best model among the result ones. Option `Number of repeats` in parameters file tell GADMA to execute optimization the necessary number of times. Moreover there are another option `Number of processes` allows GADMA to parallelize those runs.

Note: GADMA parallelize exactly several runs, not every of them. So if one ask to repeat optimization 2 times and parallelize in more than 2 processes, only two processes will be used eventually.

Recomendation: `Number of processes` should be lesser than `Number of repeats` and it will be better if it is aliquot to the `Number of repeats`.

Note: `Number of processes` shouldn't be greater than number of kernels on one's computer. Otherwise there will be no sence in parallelization.

```
# params file
...
Number of repeats : 6
Number of processes : 2 # or 3 or 6
...
```

8 Output

GADMA put all files to the directory that user set through `-o/--output` command-line option:

```
$ gadma -o output_dir -i input_fs.fs
```

or through `Output directory` option in parameters file:

```
# params file

Output directory : output_dir
...
```

8.1 Stdout and log file

GADMA prints its progress about every minute in stdout and in `output_dir/GADMA.log` file:

```
[hhh:mm:ss]
All best logLL models:
GA number       logLL               BIC               Model

All best BIC models:
GA number       logLL               BIC               Model

--Best model by log likelihood--
Log likelihood:         Best logLL
with BIC score:         BIC_score
Model:  representation of best logLL model

--Best model by BIC score--
Log likelihood:         logLL
with BIC score:         Best BIC score
Model:  representation of best BIC model
```

Note: One can set `Silence` option in parameters file to `True` to disable output in stdout, file `output_dir/GADMA.log` will still have it.

8.2 Model representation

Every model is printed as a line of parameters. All model parameters, except mutation rates, have precision equals to 2.

Consider designations:

- `T` — time,
- `P` — percent of split,
- `Ni` — size of population number `i`,
- `di` — dynamic of changing of the size of population number `i`,
- `mij` — mutation rate from population `i` to population `j`.

Dynamic of population size change has numerical values:

- `di = 0` — sudden change of size for population number `i`;
- `di = 1` — linear change of size for population number `i`;
- `di = 2` — exponential change of size for population number `i`;

Model is printed as sequence of time intervals and splits that are represented the following way:

- First period (`NA` — size of ancestry population):

  `[ NA ]`

- Split:
  - If there are one population before split event, so there will be two populations after it:

    `[ P%, [N1, N2] ]`
  - If there are two population before split event, so last formed population is divided and there will be three populations after it:

    `[ P%, [N1, N2, N3] ]`

- Usual time period:
  - If there is one population:

    `[ T, [N1], [d1] ]`
  - If there are two populations:

    `[ T, [N1, N2], [d1, d2], [[None, m12], [m21, None]] ]`
  - If there are three populations:

    ```
    [ T, [N1, N2, N3], [d1, d2, d3],
              [[None, m12, m13], [m21, None, m23], [m31, m32, None]] ]
    ```

At the end of the string, that corresponds to the model, there is information about model's ancestry in the genetic algorithm:

- `c` — for model, that is child of crossover,
- `r` — if it was formed random way,
- `m` — if it was mutated,
- `f` — final model of genetic algorithm.

Note: `m` is added as many times as model was mutated.
Example of the demographic model for two populations:

```
[ [144.38] ][ 16.00%, [23.10, 121.28] ][ 375.77, [143.31, 30.07],
                              [2, 2], [[None, 3.33e-03][7.53e-04, None]] ]
```

8.3 Output directory content

For every repeat of Genetic algorithm GADMA create new folder in output directory with correspond number.

In every folder there is `GADMA_GA.log`, where every iteration of algorithm is saved, folders `pictures` and `python_code`.

When genetic algorithm finishes GADMA saves picture and python code of result model in the corresponding folder.

When all GA finishes picture and code of best model among them are saved in root directory.

```
- <output_dir>
        - 1
                GADMA_GA.log
                - pictures
                - python_code
                        - dadi
                        - moments
                result_model.png
                result_model_code.py
        - 2
                GADMA_GA.log
                - pictures
                - python_code
                        - dadi
                        - moments
                result_model.png
                result_model_code.py
        params
        extra_params
        GADMA.log
        best_logLL_model.png
        best_logLL_model_dadi_code.py
        best_logLL_model_moments_code.py
        best_bic_model.png
        best_bic_model_dadi_code.py
        best_bic_model_moments_code.py
```

8.4 Generated code of models

By default GADMA generates python code only for final models both for ∂a∂i and *moments*. However it can do it every N iteration of genetic algorithm also. In this case option should be set in the parameters file. GADMA save files with code to the output_dir/<GA_number>/python_code directory. Both ∂a∂i and *moments* code are generated and saved in different folders there.

Each code contain function of the model, which takes values of the parameters as input, and strings that load observed AFS, simulate expected AFS from the model's function and calculate log likelihood between two AFS'. The result log likelihood is printed to stdout. For the *moments* code picture is also draw.

All code can be run the following way:

```
$ python file_with_code.py
```

Example of generated code one can find at the end of this manual in the corresponding section.

9 Plotting model

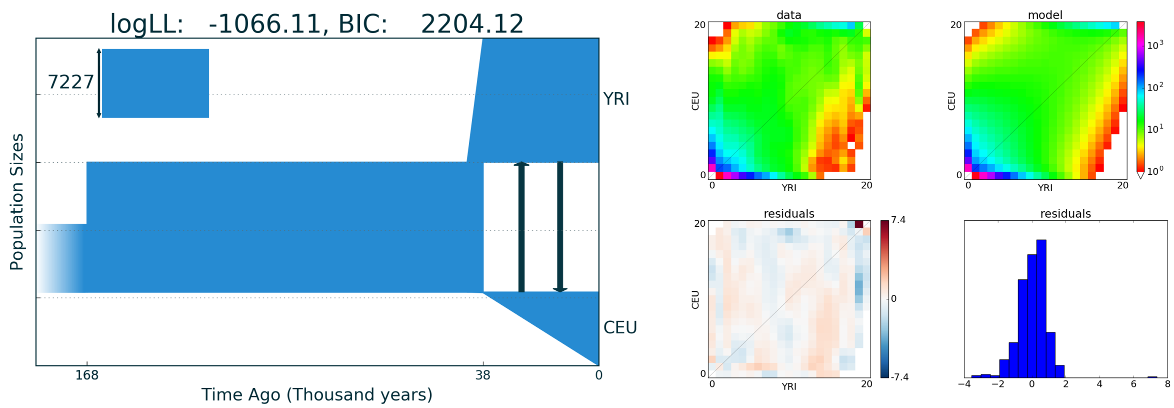

Figure 3: Exaple of demographic model plot that GADMA draw during run.

GADMA always draws final best models during genetic algorithms search for best solutions.

However models can be drawing during the pipeline every N's iteration of each genetic algorithm. N is equal to the value of `Draw models every N iteration` option in parameters file. So to enable drawing one should set this option in file.

Recommendation: Don't draw models very often, because changes are usually not very significant and drawing takes some time, so optimization will be slower.

Note: One can disable drawing by setting `Draw models every N iteration : 0` in the parameters file. This is also default behaviour.

Models are drawn with *moments* library, so it should be installed if one want to have pictures. In the top left corner there is size of ancestry population. Other parameters one can find in string representation of the model in the log files.

9.1 Time units on model's plot

Time on the demographic model's plot can be drawn in units of years, thousand years or in genetic units. By default choice depends on the `Time for generation` option in the parameters file: if it is set to some value (in years) then time will be shown in years, otherwise it will be in genetic units.

But, of course, it is possible to tell GADMA what units are preferable. For example, if one wants time to be in thousand of years on the pictures, as it is big values in years:

```
# params file
...
Units of time in drawing : thousand years
...
```

9.2 Plotting after GADMA finished

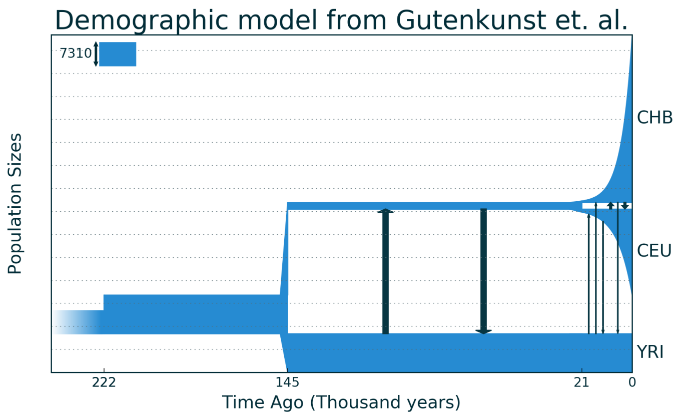

Figure 4: Example of model's plot that was drawn with generated python script.

Sometimes final pictures aren't satisfying or one didn't draw for some reasons (don't want to install *moments*, want fast launch) it is possible to plot demographic model (and only it) again.

To do it one should run corresponding generated python script for the model. For example, final model can be draw again the following way:

```
$ python best_logll_model_moments_code.py
```

Note: Again it's possible only if one have *moments* installed.

Note: One can change code inside file and draw again if picture isn't satisfying.

10 Usefull options

10.1 Time for generation

Option `Time for generation` in parameters file is corresponding to the time per one generation in Wright-Fisher model. It is responsible basically for one thing: time on the model's plots. If it is specified, then time on the pictures will be converted from genetic units by scaling with this value. Otherwise, if it is not set, time will be shown in genetic units.

Note: If `Time for generation` is specified, it should be consistent with another option: `Theta0`, which is described in the corresponding section.

10.2 `Theta0`

`Theta0` is equal to the expected number of mutation that occur in one chromosome in one generation in the infinite-sited model. GADMA can scale all values of demographic model parameters due to known value of `Theta0`. However, it is not always possible to find it. There is a way to solve this problem: one can to set `Theta0` to `None` or just not to specify it at all, so GADMA will take it as 1.0 and after launch one can scale result values due to found `Theta0` (how to do it is described in the corresponding section).

10.2.1 Estimating `Theta0`

If μ — is the neutral mutation rate per site per generation and L — length of the sequence, then:

$$\theta_0 = 4 * \mu * L$$

Note: L is the effective sequenced length, which accounts for losses in alignment and missed calls.

Note: μ **should be estimated based on generation time**. One can leave `Time per generation` option in parameters file unspecified (then time on the model's plots will be in the genetic units), but don't forget to recalculate μ!

For example (Gutenkunst et al, 2009 [1]):

> We estimate the neutral mutation rate μ using the divergence between human and chimp. Comparing aligned sequences in our data, we estimate the divergence to be 1.13%. Assuming a divergence time of 6 million years and a mean generation time for human and chimp over this interval of 25 years, we have

$$\mu = 0.0113 \cdot 25/(2 \cdot 6 \cdot 10^6) = 2.35 \times 10^{-8} \text{ per generation.}$$

10.2.2 Changing `Theta0`

If GADMA was launched with one `Theta0` and now one want to use another or if it was launched with default `Theta0` = 1.0 and now one have estimated its real value, model's parameters can be simply scaled:

Let $a = \frac{\theta_0^{NEW}}{\theta_0^{OLD}}$,

- Size of population $/a$,
- Time $/a$,
- Migration rates $*a$,
- Split percent stay the same.

10.3 Examples of `Theta0` and time for generation

The following tables produces different possible values for the demographic model inference for three populations of modern people: YRI, CEU, CHB.

FS filename	Gen. time (years)	μ (per site per gen.)	L (base pair)	`Theta0` (per chr. per gen.)
YRI_CEU_CHB.fs	25 [1]	$2.35 \cdot 10^{-8}$ [1]	$4.04 \cdot 10^6$ [1]	0.37976
YRI_CEU_CHB.fs	24 [3]	$2.26 \cdot 10^{-8}$ [1]	$4.04 \cdot 10^6$ [1]	0.36521
YRI_CEU_CHB.fs	29 [2]	$1.44 \cdot 10^{-8}$ [2]	$4.04 \cdot 10^6$ [1]	0.23270
YRI_CEU_CHB.fs	24 [3]	$1.2 \cdot 10^{-8}$ [2]	$4.04 \cdot 10^6$ [1]	0.19392

Table 1: Examples of different values of generation time and its influence on μ and `Theta0`

In Gutenkunst et al. [1] generation time for human populations was equal to 25 years and mutation rate μ was estimated as $2.35 \cdot 10^{-8}$. If one want to change time for one generation to 24 years, one need to scale μ: $\mu/25 \cdot 24 = 2.26 \cdot 10^{-8}$.

In Jouganous et al. [2] generation time was grater — 29 years and mutation rate was equal to $1.44 \cdot 10^{-8}$. To change generation time to 24, one need to change value of mutation rate: $\mu_{NEW} = \mu/29 \cdot 24 = 1.2 \cdot 10^{-8}$. `Theta0` is calculated then by the formula above.

Note: One can find FS files in `fs_examples` directory.

10.4 Relative to N_A parameters

Sometimes it is more important to see parameters scaled to $N_{ref} = N_A$. To tell GADMA show models with scaled parameters, option `Relative parameters` should be set to `True`. By default, it is `False`. It is conveniently when `Theta0` is unknown.

10.5 Disable migrations

GADMA can to exclude migrations rates from optimization and consider them be equal to zero. In that case all migrations are disabled. One should set option `No migrations` to `True` in the parameters file.

10.6 Upper bound for time split

To limit time of some split one should specify option in parameters file. Splits are numerated from the most ancient. So split 1 is split that occurred with ancient population and split 2 is next division of second population (exist only for three populations). There are two appropriate options: `Upper bound of first split` and `Upper bound of second split`.

One should translate time from years into genetic units, therefore divide it by $2 \cdot T_g$, where T_g is time (in years) for one generation. For example, one want to limit last split with 2000 years. Time for one generation is estimated as 24 years, then one should specify in parameters file:

```
...
Upper bound of second split : 83.333
...
```

10.7 Resume launch

To resume interrupted launch one can use `--resume` command-line option or set `Resume from` in the parmeters file. One need to set output directory of previous run.

If neither `Output directory` or `-o/--output` aren't specified, GADMA will continue evaluation in the directory: `<previous_output_dir>_resumed`.

10.7.1 Only models

GADMA can resume launch taking final models only from previous run. This means, that it is not resumption, but run from some initial values. It is usufull, for example, when one have run GADMA with some small grid size for dadi and then want to restart it with greater number of grid points. To do so, one should set command-line option `--only_models` with `--resume` or specify `Only models` option in parameters file to `True`.

11 Extra parameters file

GADMA take extra parameters file as input. **However one probably do not need them**. Nevertheless, if one is interested, `extra_params_template` with all options and their descriptions can be found in `gadma` folder.

12 Units

GADMA shows model parameters in genetic units. To scale them from one should multiply migration rates by 2 only. Other units are as usual. In case when option `Relative parameters` is set to `True` one should first rescale from units of N_{ref}: sizes of populations and time — multiply by `NA`, migration rates — divide by `NA`.

13 Decreasing number of model's parameters

There are several ways to decrease number of demographic model's parameters in GADMA. Firstly, as it was discussed above, one can consider only sudden dynamics of population size changes (`Only sudden : True`). Second way to decrease number of parameters is so-called multinom scheme: when at every step N_A is chosen to fit the observed data best. This scheme leads to faster work of GADMA, however it is more possible to get stuck in local optima. To use it one should set option `Multinom` to `True` in parameters file. Third way is to disable migration by setting `No migrations` to `True` in parameters file.

The last and more reliable way to decrease number of model's parameters is to use GADMA optimization on the custom model written with ∂a∂i or *moments* (section Specifying model in details). However it requires some extra work for user.

14 Dependencies

The absolute dependencies for GADMA are:

- Python (2.5, 2.6, 2.7)
- NumPy ($\geq$ 1.2.0)
- Scipy ($\geq$ 0.6.0)
- ∂a∂i ($\geq$ 1.7.0) or *moments* ($\geq$ 1.0.0)

To draw demographic models install:

- matplotlib ($\geq$ 0.98.1)
- Pillow ($\geq$ 4.2.1)
- *moments* ($\geq$ 1.0.0)

15 Installation

Before GADMA installation one need either ∂a∂i or *moments* been installed.

15.1 ∂a∂i

To install ∂a∂i, go to the work directory and run:

```
$ git clone https://bitbucket.org/gutenkunstlab/dadi/
```

Then go to the `dadi` directory and install package:

```
$ cd dadi
$ python setup.py install
```

To check ∂a∂i's installation, type in python interpreter:

```
$ python
>>> import dadi
```

15.2 *moments*

To install *moments*, go to the work directory and run:

```
$ git clone https://bitbucket.org/simongravel/moments/
```

Check that Cython is installed:

```
$ python -m pip install --upgrade Cython
```

Then go to the `moments` directory and install package:

```
$ cd moments
$ python setup.py install
```

To check *moments*'s installation, run in python interpreter:

```
$ python
>>> import moments
```

15.3 Pillow

Install Pillow throgh `pip`:

```
$ python -m pip install Pillow
```

To check Pillow's installation, type in python interpreter:

```
$ python
>>> import PIL
```

15.4 GADMA

To download GADMA, go to the work directory and run:

```
$ git clone https://github.com/ctlab/GADMA
```

Then go to the `GADMA` directory and install GADMA:

```
$ cd GADMA
$ python setup.py install
```

Now one can run it like this:

```
$ gadma --help
```

15.4.1 Verifying installation

To verify installation run test:

```
$ gadma --test
```

If the installation is successful, one will find the following information at the end:

```
--Finish pipeline--

--Test passed correctly--
Thank you for using GADMA!
```

Appendices

A Example of full parameters file

```
#       It is parameters file for GADMA software.

#   Lines started from # are ignored.
#   Also comments at the end of a line are ignored too.
#   Every line contains: Identificator of parameter : value.
#   If one want to change some default parameters, one need to
#   remove # at the begining of line and change corresponding
#   parameter.


#   Output directory to write all GADMA out.
#   One need to set it to not existing or empty directory.
#   If it is resumed from other directory and output directory
#   isn't set, GADMA will add '_resumed' for previous output
#   directory.
Output directory : my_example_run


#   One can resume from some other launch of GADMA by setting
#   output directory of it to 'Resume from' parameter.
#   You can set again new parameters of resumed launch.
#Resume from : another_output_dir
#
#       If you want to take only models from previous run set this
#       flag.  Then iterations of GA will start from 0 and values of
#       mutation rate and strength will be initial.
#       Default: None
#Only models : None


#   Input file can be sfs file (should end with .fs) or
#   file of SNP's in dadi format (should end with .txt).
Input file : fs_examples/YRI_CEU.fs

#   'Population labels' is sequence of population names (the same
#   names as in input file)
#   If .fs file is in old format then it would rename population
#   labels that are absent.
#   It is necessary to put them in order from most ancient to less.
```

```
#   (In case of more than two populations)
#   It is important, because the last of formed populations take
#   part in next split.
#   For example, if we have YRI - African population,
#   CEU - European population and CHB - Chineese populaion,
#   then we can write YRI, CEU, CHB or YRI, CHB, CEU
#   (YRI must be at the first place)
#   Default: from input file
Population labels : CEU, YRI # we change populations order
                             # (in input file YRI is first)

#   Also one can project your spectrum to less size.
#   For example, we have 80 individuals in each of three
#   populations, then spectrum will be 81x81x81 and one can
#   project it to 21x21x21 by set 'Projections' parameter
#   to 20, 20, 20.
#   Default: from input file
Projections : None # will be 20, 20




#   Now all main parameters:
#
#   Total mutation flux - theta.
#   It is equal to:
#   theta = 4 * mu * L
#   where mu - mutation rate per site per generation and
#   L - effective sequenced length, which accounts for losses
#   in alignment and missed calls.
#   Note: one should estimate mu based on generation time.
#   Default: 1.0
Theta0 : 0.37976 # the same as in Gutenkunst et al 2009

#   Time (years) for one generation. Can be float.
#   Is important for drawing models. If one don't want to draw,
#   one can pass it.
#   Default: 1.0
Time for generation : 25 # the same as in Gutenkunst et al 2009


#   Parameters for demographic models:
#
#   Use moments or dadi
#   Default: dadi
```

```
Use moments or dadi : moments

#   Use multinom scheme: N_A is not parameter for search,
#   it is calculated through optimal_sfs_scaling.
#   Multinom scheme decrease number of parameter by one and
#   is usually faster, however non multinom scheme usually
#   finds better solutions.
#   Default: False
Multinom : False

#   If you choose to use dadi, please set pts parameter - number
#   of points in grid
#   Default: Let n = max number of individuals in one population,
#   then pts = n, n+10, n+20
#Pts : 20, 30, 40

#   Structure of model for one population - number of time periods
#   (e.g. 5).
#   Structure of model for two populations - number of time periods
#   before split of ancestral population and after it (e.g. 2,2).
#   Structure of model for three populations - number of time periods
#   before first split, between first and second splits and after
#   second split (e.g. 2,1,2).
#
#   Structure of initial model:
#   Default: all is ones - 1 or 1,1 or 1,1,1
Initial structure : 2,1

#   Structure of final model:
#   Default: equals to initial structure
#Final structure : 2,2

#   It is possible to limit time of splits.
#   Split 1 is the most ancient split.
#   !Note that time is in genetic units (2 * time for 1 generation):
#   e.g. we want to limit by 150 kya, time for one generation is
#   25 years, then bound will be 150000 / (2*25) = 3000.
#
#   Upper bound for split 1 (in case of 2 or 3 populations).
#   Default: None
#Upper bound of first split : None

#   Upper bound for split 2 (in case of 3 populations).
#   Default: None
#Upper bound of second split : None
```

```
#       Print parameters of model in units of N_ref = N_A.
#   N_A will be placed in brackets at the end of string.
#   Default: False
Relative parameters : False

#   Disable migrations in demographic models.
#   Default: False
No migrations : false




#   Parameters for Genetic Algorithm.
#
#   Size of population of demographic models in GA:
#   Default: 10
Size of population in GA : 10

#   Fractions of current models, mutated models and crossed models
#   to be taken to new population.
#   Sum of fractions should be <= 1, the remaining fraction is
#   fraction of random models.
#   Default: 0.2,0.3,0.3
#Fractions in GA : 0.2,0.3,0.3

#   Mutation strength - fraction of parameters in model to mutate
#   during global mutation process of model.
#   Number of parameters to mutate is sampled from binomial
#   distribution, so we need to set mean.
#   Default: 0.2
Mean mutation strength : 0.3
#
#   Mutation strength can be adaptive: if mutation is good,
#   i.e. has the best fitness function (log likelihood),
#   then mutation strength is increased multiplying by const
#   otherwise it decreases dividing by (1/4)^const.
#   When const is 1.0 it is not adaptive.
#   Default: 1.0
Const for mutation strength : 1.05

#   Mutation rate - fraction of any parameter to change during
#   its mutation.
#   Mutation rate is sampled from truncated normal distribution,
#   so we need mean (std can be specified in extra params).
```

```
#   Default 0.2
Mean mutation rate : 0.1
#
#   Mutation rate also can be adaptive as mutation strength.
#   Default: 1.02
Const for mutation rate : 1.01 #very small changes

#   Genetic algorithm stops when it couldn't improve model by
#   more that epsilon in logLL
#   Default: 1e-2
Epsilon : 1e-2
#
#   and it happens during N iterations:
#   Default: 100
Stop iteration : 50


#   Local optimization.
#
#   Choice of local optimization, that is launched after
#   each genetic algorithm.
#   Choices:
#
#   *   optimize (BFGS method)
#
#   *   optimiza_log (BFGS method)
#
#   *   optimize_powell (Powell's conjugate direction method)
#       (Note: is implemented in moments: one need to have moments
#   installed.)
#
#   (If optimizations are often hitting the parameter bounds,
#   try using these methods:)
#   *   optimize_lbfgsb
#   *   optimize_log_lbfgsb
#   (Note that it is probably best to start with the vanilla BFGS
#   methods, because the L-BFGS-B methods will always try parameter
#   values at the bounds during the search.
#   This can dramatically slow model fitting.)
#
#   *   optimize_log_fmin (simplex (a.k.a. amoeba) method)
#
#   *   hill_climbing
#
#   Default: optimize_powell
```

```
Name of local optimization : optimize_log


#   Parameters of pipeline
#
#   One can automatically draw models every N iteration.
#   If 0 then never.
#   Pictures are saved in GA's directory in picture folder.
#   Default: 0
Draw models every N iteration : 100

#   One can automatically generate dadi and moments code for models.
#   If 0 then only current best model will be printed in GA's
#   working directory.
#   Also result model will be saved there.
#   If specified (not 0) then every N iteration model will be saved
#   in python code folder.
#   Default: 0
Print models' code every N iteration : 100

#   One can choose time units in models' plots: years or thousand
#   years (kya, KYA). If time for one generation isn't specified
#   then time is in genetic units.
#   Default: years
Units of time in drawing : thousand years

#   No std output.
#   Default: False
Silence : False

#   How many times launch GADMA with this parameters.
#   Default: 1
Number of repeats : 3

#   How many processes to use for this repeats.
#   Note that one repeat isn't parallelized, so increasing number
#   of processes doesn't effect on time of one repeat.
#   It is desirable that the number of repeats is a multiple of
#   the number of processes.
#   Default: 1
Number of processes : 3
```

B Example of generated code

For example, *moments* generated code for 2d AFS for YRI, CEU populations from [1]:

```
#current best params = [7194.792822462478, 13410.251542201073, 0.95825445
import matplotlib
matplotlib.use("Agg")
import moments
import numpy as np
def generated_model(params, ns):
        Ns = params[:5]
        Ts = params[5:7]
        Ms = params[7:]
        theta1 = 0.37976
        sts = moments.LinearSystem_1D.steady_state_1D(sum(ns),
                theta=theta1, N=Ns[0])
        fs = moments.Spectrum(sts)

        before = [Ns[0]]
        T = Ts[0]
        after = Ns[1:2]
        growth_funcs = [lambda t: after[0]]
        list_growth_funcs = lambda t: [ f(t) for f in growth_funcs]
        fs.integrate(Npop=list_growth_funcs, tf=T, dt_fac=0.1,
                theta=theta1)

        before = after
        fs = moments.Manips.split_1D_to_2D(fs, ns[0], sum(ns[1:]))

        before.append((1 - Ns[2]) * before[-1])
        before[-2] *= Ns[2]
        T = Ts[1]
        after = Ns[3:5]
        growth_funcs = [lambda t: after[0],
                lambda t: before[1] * (after[1] / before[1]) ** (t / T)]
        list_growth_funcs = lambda t: [ f(t) for f in growth_funcs]
        m = np.array([[0, params[7]],[params[8], 0]])
        fs.integrate(Npop=list_growth_funcs, tf=T, m=m, dt_fac=0.1,
                theta=theta1)

        before = after
        return fs
data = moments.Spectrum.from_file('data/YRI_CEU.fs')

popt = [7194.792822462478, 13410.251542201073, 0.9582544565961783,
```

```
        13542.979276844108, 12114.968575519626, 2683.3787253409746,
        846.6668954957415, 0.00014172779289593632, 0.00012195685425105608]
ns = [20, 20]
model = generated_model(popt, ns)
ll_model = moments.Inference.ll(model, data)
print('Model log likelihood (LL(model, data)): {0}'.format(ll_model))
#now we need to norm vector of params so that first value is 1:
popt_norm = [1.0, 1.8638829321580523, 0.9582544565961783,
            1.8823306815120924, 1.6838523185401713, 0.3729612223111333,
            0.1176777311575127, 1.0197021070711312, 0.8774542996156008]
print('Drawing model to model_from_GADMA_from_simple.png')
model = moments.ModelPlot.generate_model(generated_model, popt_norm, ns)
moments.ModelPlot.plot_model(model,
        save_file='model_from_GADMA_from_simple.png',
        fig_title='',
        pop_labels=['YRI', 'CEU'],
        nref=7194,
        draw_scale=True,
        gen_time=0.025,
        gen_time_units="Thousand years",
        reverse_timeline=True)
```

References

[1] Ryan N Gutenkunst, Ryan D Hernandez, Scott H Williamson, and Carlos D Bustamante, *Inferring the joint demographic history of multiple populations from multidimensional snp frequency data*, PLoS genetics **5** (2009), no. 10, e1000695.

[2] Julien Jouganous, Will Long, Aaron P. Ragsdale, and Simon Gravel, *Inferring the joint demographic history of multiple populations: Beyond the diffusion approximation*, Genetics **206** (2017), no. 3, 1549–1567.

[3] Marguerite Lapierre, Amaury Lambert, and Guillaume Achaz, *Accuracy of demographic inferences from the site frequency spectrum: the case of the yoruba population*, Genetics (2017), genetics–116.