`polyDFEv2.0`
User Manual

Paula Tataru

October 3, 2018

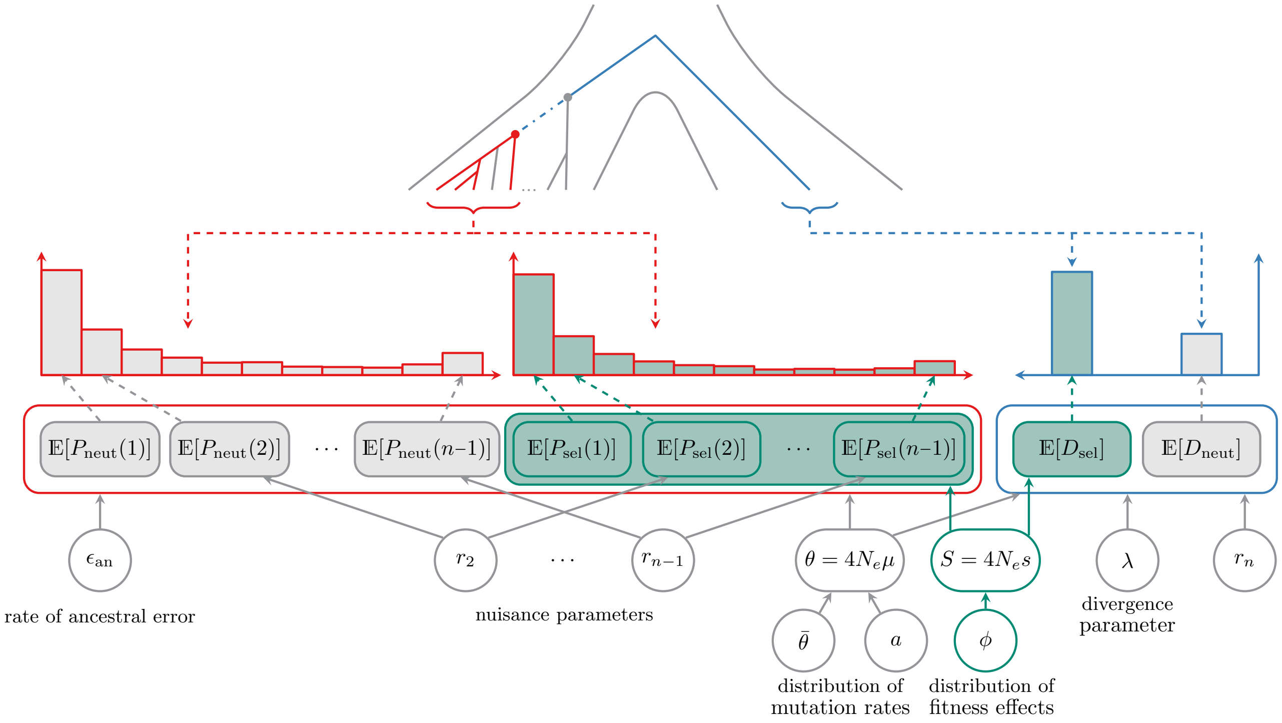
`polyDFE` infers the distribution of fitness effects (DFE) of new mutations from polymorphism, and, if available, can also incorporate divergence data obtained from an outgroup. The polymorphism data is provided as an unfolded site frequency spectrum (SFS). Note that, in order to obtain an unfolded SFS, at least one outgroup is needed. However, including the divergence data when inferring the DFE can lead to bias in the estimated parameters (see Tataru et al. (2017) for further discussion).

Once the DFE is obtained, `polyDFE` also calculates α, the rate of adaptive molecular evolution, commonly defined as the proportion of fixed adaptive mutations among all non-synonymous substitutions.

`polyDFEv2.0` (Tataru and Bataillon, 2018) can fit jointly multiple datasets for which some parameters can be set to be invariant (shared across the datasets). This enables model testing for parameter invariance, for example, testing if different datasets have the same DFE or not.

This code implements the program described in Tataru et al. (2017); Tataru and Bataillon (2018).

This manual is supplemented by a tutorial providing the full analysis of a dataset. Note that the tutorial has been made using `polyDFEv1.1` and the `R` scripts have to be slightly adapted to work with `polyDFEv2.0`.


...
$\mathbb{E}[P_{\text{neut}}(1)]$
$\mathbb{E}[P_{\text{neut}}(2)]$
$\cdots$
$\mathbb{E}[P_{\text{neut}}(n{-}1)]$
$\mathbb{E}[P_{\text{sel}}(1)]$
$\mathbb{E}[P_{\text{sel}}(2)]$
$\cdots$
$\mathbb{E}[P_{\text{sel}}(n{-}1)]$
$\mathbb{E}[D_{\text{sel}}]$
$\mathbb{E}[D_{\text{neut}}]$
ϵ_{an}
rate of ancestral error
r_2
$\cdots$
r_{n-1}
nuisance parameters
$\theta = 4N_e\mu$
$S = 4N_e s$
λ
r_n
divergence parameter
$\bar{\theta}$
a
ϕ
distribution of mutation rates
distribution of fitness effects

Contents

1 Authors

`polyDFE` has been developed and implemented by Paula Tataru and Marco A.P. Franco.

2 License

`polyDFE` is open source, distributed under the GNU General Public License, version 3. See `LICENSE.txt` for details.

3 Requirements

`polyDFE` is implemented in C and uses the GSL library. This should be downloaded from https://www.gnu.org/software/gsl/ and compiled accordingly. `polyDFE` has been tested using GSL-1.16. Note that `polyDFE` does not work with a newer version of GSL.

4 Installation

The simplest way to compile `polyDFE` is

1. `cd` to the directory containing the `makefile` and type

   ```
   make all
   ```

   to compile the code.

2. You can remove the program binaries and object files from the source code directory by typing

   ```
   make clean
   ```

If the `GSL` library is not installed in the default directory, than the installation path needs to be specified in the `makefile` by updating `USR_INC` and `USR_LIB` and uncommenting the lines at the top of the `makefile`:

```
############################################################
# for non-default GSL installation
############################################################
# USR_INC := -I<ROOT PATH TO GSL 1.16>/include/
# USR_LIB := -L<ROOT PATH TO GSL 1.16>/lib/
############################################################
```

`polyDFEv2.0` is also distributed as pre-compiled binaries for Windows, Linux and macOS. When using Linux and macOS, it might be necessary to change the access permissions to the binary in order to execute it. This can be achieved by

```
chmod +x polyDFE-2.0-<os>
```

where `polyDFE-2.0-<os>` is the `polyDFEv2.0` binary.

5 New in `v2.0`

`polyDFEv2.0` enables model testing across different data sets: often when inferring the DFE, the final aim is to test if the DFE differs significantly across different areas of the genome or different species. `polyDFEv2.0` can estimate parameters jointly over multiple data sets (see -d), where some parameters (for example, the DFE) are shared across the data sets, while the rest of the parameters are estimated independently for each data set (see -i, -r and -g). Additionally, `polyDFEv2.0` implements a new optimization method (see -o).

`postprocessing.R` has also been updated to handle the new output format of `polyDFEv2.0`.

6 Error reporting

`polyDFE` has in-built checks which, if they fail, they stop the execution and print an error message. If this or any other type of problematic behaviour is encountered, please report it directly on https://github.com/paula-tataru/polyDFE or by sending an email to paula.tataru at gmail.com.

7 Running polyDFE

Running polyDFE with the argument -h will print out the usage (required and optional arguments):

```
$ ./polyDFE -h
./polyDFEv2.0
Usage: ./polyDFE -d data_file_1[:data_file_2:...:data_file_j]
        [-m model(A, B, C, D) [K]] [-i init_file ID_1[:ID_2:...:ID_j]] [-t]
        {-s m_neut L_neut m_sel L_sel n ||
         [-o optim(bfgs, conj_pr, conj_fr, simplex)] [-k kind(s, j, s+j)]
               [-r range_file ID_1[:ID_2:...:ID_j]] [-i init_file ID [-j]]
               [-e] [-w] [-g grouping_file ID_1[:ID_2:...:ID_j]] [-p optim_file ID]
               [-b [basinhop_file ID]] [-l min] [-v verbose(0, 1, frequency)]}
```

polyDFE runs in two modes: -s, used for simulating data under the hierarchical probabilistic model, and -o, used for estimating the DFE and α. If neither -s or -o are given, then -o bfgs is used by default (see -o for details). If both -s and -o are given, polyDFE will terminate with an error. Optional arguments are given between [].

7.1 Specifying the data file(s)

Regardless of the mode polyDFE is ran in, the files containing the data have to be specified using the following argument:

- -d data_file_1[:data_file_2:...:data_file_j]: path to one or multiple data files.

  data_file_i, for $1 \leq i \leq j$ is where polyDFE will write the simulated data, if -s is used, or will read the input data, if -o is used.

  The files can contain any number of empty lines and comment lines that start with #. Excluding these, the structure is

```
1                 m_neut  m_sel  n
2                 p_neut(1)  p_neut(2)  ...  p_neut(n-1)   l_neut    d_neut  l_d,neut
...               ...
m_neut+1          p_neut(1)  p_neut(2)  ...  p_neut(n-1)   l_neut    d_neut  l_d,neut
m_neut+2          p_sel(1)   p_sel(2)   ...  p_sel(n-1)    l_sel     d_sel   l_d,sel
...               ...
m_neut+m_sel+1    p_sel(1)   p_sel(2)   ...  p_sel(n-1)    l_sel     d_sel   l_d,sel
```

  where

  - m_{neut} and m_{sel} are the number of fragments that are assumed to be evolving neutrally or under selection, respectively;
  - n is the ingroup sample size (number of sequences / haplotypes);
  - $p_z(i)$ is the i^{th} entry in the SFS within the fragment, for sites that are assumed to be evolving neutrally, $z = \text{neut}$, or under selection, $z = \text{sel}$;
  - l_z is the total number of sites within the fragment used for the SFS (number of successfully sequenced sites within the ingroup) that are assumed to be evolving neutrally, $z = \text{neut}$, or under selection, $z = \text{sel}$;
  - d_z is the total number of fixed mutations in the ingroup relative to the outgroup (divergence counts) within the fragment, that are assumed to be evolving neutrally, $z = \text{neut}$, or under selection, $z = \text{sel}$;
  - $l_{\text{d},z}$ is the total number of sites within the fragment used for the divergence counts (number of successfully sequenced sites within the ingroup and outgroup) that are assumed to be evolving neutrally, $z = \text{neut}$, or under selection, $z = \text{sel}$; for a given fragment, l_z and $l_{\text{d},z}$ could be different if certain sites are successfully sequenced in the ingroup, but not in the outgroup.

  The divergence data (last two columns, d_z and $l_{\text{d},z}$) can be absent from the file. If it is absent, it should be absent for all fragments.

  polyDFE uses data divided into fragments in order to model variability in mutation rates, as described in Tataru et al. (2017). If only one neutral and one selected fragments are provided, than variability in mutation rates is not modelled.

The provided files `input/example_1`, `input/example_2` and `input/example_3` are examples of input data.

7.2 Specifying the DFE and parameters of interest

Regardless of the mode `polyDFE` is ran in, the assumed DFE model has to be specified using the following argument:

- `-m model(A, B, C, D) [K]`: assumed DFE model.

  `polyDFE` assumes that the DFE ϕ takes one of four specific functional forms, encoded `A, B, C,` and `D`, described below

  - `A`: the DFE is given by a reflected displaced Γ distribution, parameterized by $\bar{S}$, b and $S_{\max}$, with density

    $$\phi_A(S;\bar{S},b,S_{\max}) = \begin{cases} f_\Gamma(S_{\max}-S;S_{\max}-\bar{S},b) & \text{if } S \le S_{\max} \\ 0 & \text{otherwise} \end{cases}$$

    where $\bar{S}$ is the mean of the DFE, b is the shape of the Γ distribution, $S_{\max}$ is the maximum value that S can take, and $f_\Gamma(x;m,b)$ is the density of the Γ distribution with mean m and shape b.

  - `B`: the DFE is given by a mixture of a Γ and discrete distributions, parameterized by S_d, b, p_b and S_b, where

    $$\phi_B(S;S_d,b,p_b,S_b) = \begin{cases} (1-p_b)f_\Gamma(-S;-S_d,b) & \text{if } S \le 0 \\ p_b & \text{if } S = S_b \\ 0 & \text{otherwise} \end{cases}$$

    where S_d is the mean of the DFE for $S \le 0$, b is the shape of the Γ distribution, p_b is the probability that $S > 0$, and S_b is the shared selection coefficient of all positively selected mutations, and $f_\Gamma(x;m,b)$ is the density of the Γ distribution with mean m and shape b.

  - `C`: the DFE is given by a mixture of a Γ and Exponential distributions, parameterized by S_d, b, p_b and S_b, where

    $$\phi_C(S;S_d,b,p_b,S_b) = \begin{cases} (1-p_b)f_\Gamma(-S;-S_d,b) & \text{if } S \le 0 \\ p_b\, f_e(S;S_b) & \text{if } S > 0 \end{cases}$$

    where S_d is the mean of the DFE for $S \le 0$, b is the shape of the Γ distribution, p_b is the probability that $S > 0$, S_b is the mean of the DFE for $S > 0$, and $f_\Gamma(x;m,b)$ is the density of the Γ distribution with mean m and shape b, while $f_e(x;m)$ is the density of the Exponential distribution with mean m.

  - `D`: the DFE is given as a discrete DFE, where the selection coefficients can take one of S_i distinct values, $1 \le i \le$ `K`, where each value S_i has probability p_i, with

    $$\sum_{i=0}^{K} p_i = 1$$

    The value of `K` needs to be provided in the command line for model `D`.

  If `-m` is not given, model `C` is used by default.

When simulating data (mode `-s`), the DFE and other additional parameters have to be specified using `-i`. This can also optionally be used when inferring the parameters (mode `-o`). This way, the user can control which parameters should be estimated and which should be kept fixed to a given value, and can also provide the initial values of the parameters used during the optimization. Note that if using `-m D`, `-i` is no longer optional, as it contains the information about the selection coefficients S_i.

- `-i init_file ID_1[:ID_2:...:ID_j]`: path to a file containing the values of the DFE and other parameters, and one or multiple IDs.

  `init_file` can contain any number of empty lines and comment lines that start with `#`. Excluding these, the structure of each line is `id [0|1|2]` ϵ_{an} `[0|1|2]` ϵ_{cont} `[0|1|2]` λ `[0|1|2]` $\bar{\theta}$ `[0|1|2] a <DFE parameters> [0|1|2]` $\mathtt{r}_2..\mathtt{r}_n$

Each line starts with a positive numerical id. This is used such that multiple configurations of the parameters can be stored in the same file. When running `polyDFE`, the desired line for initializing the parameters is specified through the id by setting `ID_i`, for $1 \leq i \leq j$, to the right value.

The id is followed by the values of the parameters, in the order ϵ_{an}, ϵ_{cont}, λ, $\bar{\theta}$, a, the DFE parameters (see `-m`), and the r_i, $2 \leq i \leq n$, parameters, where n is the ingroup sample size. The r_i parameters can also be provided for $i > n$, but those values will be ignored. This is useful if the same line in `init_file` is used for multiple data sets that have different sample sizes.

All parameters are preceded by a flag, which is either 0, 1 or 2. In the `-s` mode, the flag is not used, but this becomes important for the `-o` mode, where the flag indicates if a parameter should be estimated independently (flag 0), should be kept fixed to the value provided in `init_file` (flag 1), or should be estimated but shared across the multiple input data files (see `-d`). All r_i parameters share one flag.

When running `polyDFE` jointly over multiple data files (see `-d`), either only one ID should be provided, which is then used for all the data files, or a number j (the same as the number of input data files) of IDs is required. In the latter case, `ID_i` will be used for `data_file_i`, for $1 \leq i \leq j$. If one parameter has flag 2 for some `ID_i` but flag 0 for some other `ID_l`, than this parameter is estimated and shared over all the data files.

When `-m D` is used, only the probabilities p_i are estimated, while the selection coefficients S_i are fixed to the given values.

For details of the meaning of each parameter, please see Tataru et al. (2017), noting that ϵ_{an} is simply referred to as ϵ.

The parameter ϵ_{cont} is deprecated since `v1.1` and is no longer used. For backward compatibility, it is still present in the `init_file`.

The a parameter is directly linked to variability of mutation rates. If it is wished for mutation rates to be assumed constant, than a can be fixed to -1. If both m_{neut} and m_{sel} provided in the input datafile (through argument `-d`) are equal to 1 (i.e. the data is not divided into fragments), then the mutation variability is not estimated by default. When mutation variability is allowed, than the observed data (counts) are assumed to follow a negative binomial distribution, otherwise they are assumed to follow a Poisson distribution. See Tataru et al. (2017) for more details on mutation variability. Note that if the variability in mutation rate is not of interest, then a can be fixed to -1 without biasing the inference of the remaining parameters.

The provided files `input/init_model_A.txt`, `input/init_model_BandC.txt` and `input/init_model_D.txt` give examples of the structure of the `init_file` for the different DFE models.

7.3 Simulating data

The simulation mode (`-s`) requires the following argument:

- `-s m_neut L_neut m_sel L_sel n`: simulate data.

  When using `-s`, simulated data will be printed in `data_file` with m_{neut} = `m_neut`, m_{sel} = `m_sel`, $l_{\text{neut}} = l^d_{\text{neut}}$ = `L_neut` and $l_{\text{sel}} = l^d_{\text{sel}}$ = `L_sel`, for all simulated fragments. For details on m_z and l_z, with $z \in \{\text{neut}, \text{sel}\}$, see `-d`.

7.3.1 Examples

When simulating data, `polyDFE` throws a warning about overwriting the file and requires input from the user to confirm the action:

```
$ ./polyDFE -d input/example_1 -m A -s 500 2000 500 6000 20 -i input/init_model_A.txt 1
---- Running command
---- ./polyDFE -d input/example_1 -m A -s 500 2000 500 6000 20 -i input/init_model_A.txt 1
```

```
Warning: simulating data, input/example_1 will be overwritten.
Do you want to continue with simulation? (Y/N): Y
```

This is to ensure that no files containing data are overwritten by mistake.

The provided files `input/example_1`, `input/example_2` and `input/example_3` are obtained by calling

```
$ ./polyDFE -d input/example_1 -m A   -s 500   2000 500   6000 20 -i input/init_model_A.txt 1
$ ./polyDFE -d input/example_2 -m C   -s  50  20000  50  60000 20 -i input/init_model_BandC.txt 1
$ ./polyDFE -d input/example_3 -m D 5 -s 100  10000 100  30000 20 -i input/init_model_D.txt 1
```

The simulated `data_file` contains information at the beginning about the DFE model used for the simulation, together with the values of all parameters.

7.4 Estimating the DFE and α

The estimation (optimization) mode (`-o`) allows for the estimation of DFE and α. For this, next to the arguments described above, it allows for a series of optional arguments, which control the optimization, both how the parameters are estimated, but also when a set of parameters found is considered to be optimal.

7.4.1 Controlling the optimization procedure

One of the key issues in inferring the parameters is to ensure that the likelihood function is properly optimized over the space of parameters. `polyDFE` implements multiple steps to ensure, as much as possible, that good parameters are found. These can be customized by the user using the following arguments:

- `-o (bfgs, conj_pr, conj_fr, simplex)`: optimization method to use for estimating the parameters.

  `polyDFE` uses `GSL` to run a local optimization algorithm and find estimates of the parameters. `GSL` implements multiple optimization algorithms. Generally, optimization is more efficient if it also uses the derivative of the function to be optimized. Therefore, in `polyDFE` we have chosen to use three of the algorithms offered by `GSL`, which all rely on the function (in this case, the data likelihood) derivatives. Additionally, `polyDFE` also implements one method that does not rely on the derivative:

  ◦ `bfgs`: the Broyden-Fletcher-Goldfarb-Shanno (BFGS) algorithm.

  ◦ `conj_pr`: the Polak-Ribiere conjugate gradient algorithm;

  ◦ `conj_fr`: the Fletcher-Reeves conjugate gradient algorithm;

  ◦ `simplex`: the the Simplex algorithm of Nelder and Mead.

  For details on the different optimization algorithms, please see the `GSL` manual. When `-o` is not given, `-o bfgs` is used by default. `polyDFE` has been tested using the BFGS algorithm.

  Note that the optimization of the parameters for model `D` (see `-m` for details) is more difficult, due to the fact that the probabilities p_i are constrained to sum to 1, while all of the above algorithms are unconstrained.

- `-p optim_file ID`: path to a file containing parameters that control the optimization algorithm, and id.

  `optim_file` can contain any number of empty lines and comment lines that start with `#`. Excluding these, the structure of each line is

  `id` ϵ_{abs} `step size` `tolerance` `max iterations`

  Each line starts with a positive numerical id. This is used such that multiple configurations for the optimization algorithms can be stored in the same file. When running `polyDFE`, the desired line for parameters that control the optimization is specified through the id by setting `ID` to the right value.

  The `optim_file` is used to control the behavior of the optimization algorithms from `GSL`, specified using `-o`:

  ◦ ϵ_{abs}: this the absolute tolerance value that the gradient or the minimizer-specific characteristic size (for `-o simplex`) of the function to be optimized (here, the likelihood) is tested against. When the gradient / size is smaller than this values, than `GSL` considers that it has successfully found an optimum. If this value is too large, it will lead to an early termination of the algorithm. If the value is too small, it might lead to a longer running time of the algorithm without, potentially, much improvement on the function. However, it is always better to have an absolute tolerance value that is too small than too large. If a set of values found is a true optimum, than the gradient / size should be 0. Default value: 0.00001 and 0.01 for `-o simplex`.

  - step size: the `GSL` optimization algorithms use a line search and the step size determines the size of the first step performed in the search. Default value: 2.
  - tolerance: specifies the accuracy of the line search, with the precise meaning depending on the algorithm used. Default value: 0.1.
  - max iterations: the `GSL` optimization algorithms are iterative. If the number of performed iterations reaches max iter, than the algorithm is stopped, regardless of the gradient / size. Default value: 2000 and 20000 for `-o simplex`.

  For more details on the above, please consult the `GSL` manual.

  The provided file `input/params.optim` is an example of an `optim_file`.

- `-l min`: limit running time to `min`.

  To control the running time of `polyDFE`, one run of the optimization algorithm is stopped if it has used more than `min`. By default, `min` is set to 300 (5h). Note that if `min` is set too low, there is a high chance that the optimization algorithm will terminate without finding good optimal parameters. Setting `min` to a negative number will allow the optimization algorithm to run until it finishes, without a limited running time.

- `-e`: automatically estimate the initial value of the parameters.

  The optimization algorithms require an initial value for the parameters to be optimized. This can either be provided using `-i` if the user has a good idea for such values, or can, by using `-e`. If `-e` is used without `-i`, then all the parameters (expect for ϵ_{cont}, which, by default, is not estimated and set to 0) will be automatically initialized and then optimized. When multiple data files are provided (see `-d`), all parameters are shared (flag 2, see `-i`) by default. However, if `-i` and `-e` are used together, the initial value of the parameters is estimated automatically only for those parameters that are not flagged with 1 and are consequently optimized.

  The parameters $\bar{\theta}$, a, λ and r_i can be estimated deterministically. The rest of the parameters are estimated using a grid search approach. However, for the DFE model `D`, the probabilities p_i are set to be uniform when `-e` is used.

  If `-i` is not used, `-e` is used by default.

- `-j`: use jointly both the provided initial values and automatically estimated values of parameters.

  When `-j` is used together with `-i`, then the optimization algorithm is ran twice, once where the initial value of the parameters is the one given in the `init_file`, and a second time where the automatic estimation from `-e` is used. At the end, the parameters that have the best likelihood, found from one of the two runs, will be reported.

- `-k kind(s, j, s+j)`: kind of likelihood optimization to use for estimating the parameters.

  The likelihood computation is split in two parts: the neutral likelihood for the neutral SFS, which does not require the DFE, and the selected likelihood for the selected SFS, which requires all the parameters, including the DFE. Many DFE inference methods first infer all the parameters but the DFE (to which we refer as the neutral parameters) from the neutral SFS, and then conditional on those, they infer the DFE from the selected SFS. By default, `polyDFE` infers all of the parameters jointly by using both the neutral and selected SFS, which is computationally more costly as many parameters have to be optimized at the same time. The kind of likelihood optimization can be controlled through `-k`:

  - `s`: `polyDFE` is ran in a `single` mode, where the neutral likelihood is first optimized, followed by the optimization of the selected likelihood;
  - `j`: `polyDFE` is ran in a `joint` mode, where the joint likelihood is optimized;
  - `s+j`: `polyDFE` is ran in a `single + joint` mode, where first the `single` mode is used and the parameters found are used as initial values in the following `joint` mode.

  By default, `polyDFE` uses `-k j`. Note that using `-k s` might lead to parameters that are not optimal.

- `-r range_file ID_1[:ID_2:...:ID_j]`: path to a file containing the allowed range of all parameters, and one or multiple IDs.

  `range_file` can contain any number of empty lines and comment lines that start with `#`. Excluding these, the structure of each line is

  `id` `k` $\epsilon_{\texttt{an}}^{\texttt{min}}$ $\epsilon_{\texttt{an}}^{\texttt{max}}$ $\epsilon_{\texttt{cont}}^{\texttt{min}}$ $\epsilon_{\texttt{cont}}^{\texttt{max}}$ $\lambda^{\texttt{min}}$ $\lambda^{\texttt{max}}$ $\bar{\theta}^{\texttt{min}}$ $\bar{\theta}^{\texttt{max}}$ $a^{\texttt{min}}$ $a^{\texttt{max}}$ `<DFE parameters>` $r^{\texttt{min}}$ $r^{\texttt{max}}$

  Each line starts with a positive numerical id. This is used such that multiple configurations of the parameters can be stored in the same file. When running `polyDFE`, the desired line for the range of the parameters is specified through the id by setting `ID` to the right value.

  The id is followed by k, which controls the transformation of the parameters (see below) and the range within each parameter should be estimated. Some parameters are intrinsically constrained (for example, p_b for DFE models `B` and `C` is, by construction, constrained to be between 0 and 1), while many parameters have to be positive (or negative, for example, S_d for DFE models `B` and `C`). In order to treat all parameters in the same way and to also allow the user to narrow the range in which the optimum parameters are searched, all parameters are then constrained to the range provided by the user through the `range_file`. However, the optimization algorithms `polyDFE` uses (see `-o` for details) are unconstrained, in that the parameters can take any value in the interval $(-\infty, +\infty)$. In order to change the original constrained optimization problem to an unconstrained one, `polyDFE` transforms the parameters from the range provided by the user to the interval $(-\infty, +\infty)$. For this, a generalized logistic function is used. This function is parameterized by a growth rate (or steepness of curve) k, which controls the spread of the transformation: the smaller the k, the more quicker the transformed values move from $-\infty$ to $+\infty$. `polyDFE` has been tested with a value of $k = 0.01$. If the optimization algorithm cannot find a good solution (i.e. the gradient / size is small, see `-p` for details), the k parameter can be changed in the hope of better performance.

  Note that, in order for the optimization algorithm to successfully find the optimum parameters (the parameters that have the largest likelihood), the ranges should be large enough to ensure that they cover the optimum values. If a parameter is inferred to be very close to one of the borders of its range, than the range should be expanded.

  If initial values of the parameters provided through `-i` or as calculated when `-e` is provided are not within the ranges, the program will automatically update the ranges.

  When part of the DFE is given by a Γ distribution (see `-m`), the parameter b controls its shape. If b is very small and therefore the distribution is very leptokurtic, it is difficult to calculate the likelihood accurately, and it is thus recommended to keep the minimum range for b at least 0.01.

  If ranges are not provided, than `polyDFE` sets automatic ranges that are very large.

  The parameter ϵ_{cont} is deprecated since `v1.1` and is no longer used. For backward compatibility, it is still present in the `range_file`.

  When running `polyDFE` jointly over multiple data files (see `-d`), either only one ID should be provided, which is then used for all the data files, or a number j (the same as the number of input data files) of IDs is required. In the latter case, `ID_i` will be used for `data_file_i`, for $1 \leq i \leq j$.

  The provided files `input/range_model_A.txt`, `input/range_model_BandC.txt` and `input/range_model_D.txt` are examples of `range_file`.

- `-g grouping_file ID_1[:ID_2:...:ID_j]`: path to a file containing grouping information for the r_i parameters, and one or multiple IDs.

  `grouping_file` can contain any number of empty lines and comment lines that start with `#`. Excluding these, the structure of each line is

  `id` `G` $\texttt{i}_1$ $\texttt{i}_2$ ... $\texttt{i}_G$

  Each line starts with a positive numerical id. This is used such that multiple configurations of the grouping can be stored in the same file. When running `polyDFE`, the desired line for the range of the parameters is specified through the id by setting `ID` to the right value.

  The `grouping_file` is used to provide information on grouping the r_i parameters. By default, each entry $1 \leq i < n$ in the SFS, and the divergence count, have one value of r_i associated to it. If n is very large, this leads to a lot of parameters that need to be estimated. One could expect that the distortion that is incurred on the counts that are modeled using the r_i parameters could be similar for neighboring values (for example, $i-1$, i and $i+1$). Then only one r value can be estimated for those counts that share a similar distortion.

The grouping given in `grouping_file` will result in r values corresponding to the SFS entries as follows (where one interval represents one range for the SFS entries):

$$\begin{array}{ccccc} r_1 & r_2 & r_3 & \dots & r_{G+1} \\ [1] & [2, i_1] & [i_1+1, i_2] & \dots & [i_G, n'] \end{array}$$

where n' is either n, if divergence data is used in the estimation (see `-w` for details), or $n-1$ otherwise. For identifiability reasons, r_1 is always set to 1 and not estimated.

When running `polyDFE` jointly over multiple data files (see `-d`), either only one ID should be provided, which is then used for all the data files, or a number j (the same as the number of input data files) of IDs is required. In the latter case, `ID_i` will be used for `data_file_i`, for $1 \le i \le j$. Note that if the r parameters are shared (flag 2, see `-i`), than the grouping used has to be same for the data files.

The provided file `input/params.grouping` is an example of a `grouping_file`.

- `-b [basinhop_file ID]`: path to a file containing parameters that control the basin-hopping algorithm, and id.

  `basinhop_file` can contain any number of empty lines and comment lines that start with `#`. Excluding these, the structure of each line is

  ```
  id   max same   max iterations   temperature   step   accept rate   interval   factor
  ```

  Each line starts with a positive numerical id. This is used such that multiple configurations for the basin-hopping algorithm can be stored in the same file. When running `polyDFE`, the desired line for the range of the parameters is specified through the id by setting `ID` to the right value.

  By default, `polyDFE` runs the optimization algorithm once (or twice, see `-j` for details). However, the optimization algorithms are local, in that they only find an optimum that is in the neighborhood of the initial values of the parameters. To look more thoroughly for a global optimum, `polyDFE` allows the user to run the basin-hopping algorithm (Purisima and Scheraga, 1987; Wales and Doye, 1997; Wales and Scheraga, 1999). `polyDFE` contains a reimplementation of the basin-hopping algorithm found in the `Python` library, `scipy` (Jones et al., 2001–; Wales and Doye, 1997).

  Basin-hopping attempts to find the global optimum by iterating the following steps

  1. perturbing randomly the current value of the parameters;
  2. running the local optimization algorithm (as chosen through `-o`) using the new values as initial values;
  3. accepting or rejecting the found optimum based on the likelihood and temperature.

  Basin-hopping has been shown to be very efficient for a wide variety of problems in physics and chemistry. Unfortunately, there is no way to determine if the true global optimum has actually been found, and therefore it is left to the user to ensure that. The algorithm is controlled through

  - max same: If after max same iterations, the basin-hopping algorithm does not find an improved set of parameters, than the algorithm stops. Default value: 50.
  - max iterations: The basin-hopping algorithm is ran for a maximum number of iterations. Default value: 500.
  - temperature: The temperature is used in the metropolis criterion when accepting or rejecting the new values. For best results, the temperature should be comparable to the typical difference in likelihood between local optima. Default value: 1.
  - step: The step controls how far away from the current value the perturbated value is. This is crucial for the algorithm's performance. Ideally, it should be comparable to the typical separation between local optima of the likelihood. The algorithm implemented in `polyDFE` will automatically adjust the step, but it make take many iterations to find an optimal value. Default value: 50.
  - accept rate: The target accept rate (percentage of new values that are accepted in step 3) for when adjusting the step. Default value: 0.5.
  - interval: Every interval iterations, the basin-hopping algorithm adjusts the step. Default value: 10.
  - factor: When the step is adjusted, it is done with this factor. Default value: 0.9.

-b can be used without basinhop_file ID. If only -b is given, then polyDFE runs basin-hopping with the default parameter values given above.

The provided file input/params.basinhop is an example of a basinhop_file.

7.4.2 Excluding divergence data

By default polyDFE performs the inference using all available data, including divergence data if this is present in the data file given through -d data_file. Divergence data can be excluded from the analysis by using the argument:

- -w: do not use divergence data.

  When -w is used, then the DFE and other parameters are estimated without using divergence data, as described in Tataru et al. (2017). If the divergence data is absent from the data_file, -w will be used by default.

7.4.3 Controlling the polyDFE output

polyDFE prints to standard output information about what type of analysis it performs, status on the optimization procedure and, when this is completed, the best likelihood, gradient / size and parameters found, the expected SFS (and divergence, if -w was not used) and estimated α. The frequency of prints regarding the optimization procedure are controlled through the argument:

- -v verbose(0, 1, frequency): verbose frequency.

  By default, verbose is set to 0, and only the names of the parameters that are estimated, their initial and final values are printed. Otherwise, every verbose iterations of the optimization algorithm, information is printed about the current value of the parameters, the corresponding likelihood, gradient / size and status of the optimization. Apart from the possible status values that GSL uses, polyDFE uses the additional values:

  - -3: local optimization is stuck in the same parameter values (it cannot further improve on the current values).
  - -4: restarting the local optimization lead to the same parameter values.
  - -5: the local optimization is being restarted; sometimes, when the local optimization cannot further improve on the current values, restarting the optimization can allow it to proceed further. If the local optimization is stuck in the same values, restarting is done at most 3 consecutive times.
  - -6: the local optimization used at least min minutes given through -l, and has been stopped.
  - -7: the likelihood evaluation returned NAN and the local optimization is therefore stopped. This can happen for certain values of the parameters, where the numerical integration over the DFE fails.
  - -8: a maximum number of likelihood evaluations has been reached, and the local optimization has been stopped.

When running polyDFE on multiple data files (see -d), the names of the parameters are preceded by a flag i that indicates if this parameter is either shared ($i = 0$) or that it is estimated for data_file_i, with $1 \leq i \leq j$.

7.4.4 Additional arguments

- -t: the total running time of polyDFE will be measured and printed on the standard output at the end.

7.4.5 What to do when the gradient / size is large

If inferred parameters are at a true optimum, than the gradient / size of the corresponding likelihood should be 0. If the gradient /size is large, this indicates that the optimization failed in finding good values for the parameters. The run of polyDFE can be changed in multiple ways to try to address this problem:

- The optimization algorithm terminates with status 0 (see -v) if the gradient / size is smaller than the absolute tolerance value, ϵ_{abs}. If this is too large, the optimization algorithm might terminate prematurely. The value can be changed using -p.

- The optimization algorithm is ran for a certain number of iterations. If this number is reached, `polyDFE` prints the message "reached maximum number of iterations allowed". If this happens and the gradient / size is still large, the number of iterations should be increased using `-p`.
- The optimization algorithm has a time limit, after which it is stopped. In this case, `polyDFE` prints the message "reached maximum running time allowed". If this happens and the gradient / size is still large, the running time allowance should be increased using `-l`.
- If `polyDFE` was ran with `-k s` or `-k s+j`, setting `-k j` might lead to better estimates. See `-k` for details.
- If the inferred value of one of the parameters is very close to one of the borders of its range, than its range should be consequently increased using `-r`.
- The optimization algorithm's performance is highly dependent on the initial values of the parameters. To investigate a wider range of initial values, the basin-hopping algorithm can be ran using `-b`.
- Different optimization algorithms can lead to better parameters found. The optimization algorithm can be changed using `-o`.
- The parameters are transformed from their range to $(-\infty, +\infty)$ using a generalized logistic function which depends on a parameter k. Choosing a different value for k by using `-r` might lead to a better result.
- If the sample size n of the data is large, then a lot of parameters have to be estimated, which might interfere with the optimization process. To reduce the number of parameters, the r parameters can be grouped by using `-g`. However, an initial optimization is still needed to gain some information on how the grouping of the r parameters should be done. Then automatic grouping can be calculated in `R` using `createInitLines`.

7.4.6 Examples

Inferring a full DFE using model `C` using default arguments:

```
$ ./polyDFE -d input/example_2 > output/example_2_full_C
```

Inferring a full DFE using model `C`, while running the optimization algorithm twice, once initialized with estimated parameters, and once with parameters specified from file:

```
$ ./polyDFE -d input/example_2 -i input/init_model_BandC.txt 1 -v 20 -j > output/example_2_init_C
```

Inferring a deleterious DFE using model `C`:

```
$ ./polyDFE -d input/example_2 -i input/init_model_BandC.txt 2 -v 100 > output/example_2_del
```

Inferring a full DFE using model `C` without mutation variability:

```
$ ./polyDFE -d input/example_2 -i input/init_model_BandC.txt 3 > output/example_2_novar_C
```

Inferring a full DFE using model `C` without nuisance r parameters, and using different kinds of likelihood optimization and parametrization of the optimziation algorithm:

```
$ ./polyDFE -d input/example_2 -i input/init_model_BandC.txt 4 -j > output/example_2_nonuis_C
$ ./polyDFE -d input/example_2 -i input/init_model_BandC.txt 4 -j -k s \
           -p input/params.optim 4 > output/example_2_nonuis_s_C
$ ./polyDFE -d input/example_2 -i input/init_model_BandC.txt 4 -j -k s+j \
           > output/example_2_nonuis_sj_C
```

The above examples illustrate that using `-k s` and `-k s+j` can give poor estimates. When the runs are initialized with the values calculated automatically (see `-j`), `-k s` terminates with likelihood -3930.524, while `-k s+j` terminates with likelihood -3928.647. The default run (`-k j`) finds a much better likelihood of -3924.453. However, when the runs are initialized with the values used for simulating the data from `input/init_model_BandC.txt`, all runs find likelihoods that are approximately -3924.5. This indicates that `-k s` and `-k s+j` should be used with caution, as they might lead to suboptimal estimates.

Inferring a full DFE using model `A` and 10 basin-hopping iterations:

```
$ ./polyDFE -d input/example_2 -m A -i input/init_model_A.txt 1 -e \
           -b input/params.basinhop 1 -v 200 > output/example_2_full_A
```

Inferring a full DFE using model C where this is shared or not across 2 data files:

```
$ ./polyDFE -d input/example_1:input/example_2 -i input/init_model_BandC.txt 1 -e \
           -v 200 > output/example_1_2_indep
$ ./polyDFE -d input/example_1:input/example_2 -i input/init_model_BandC.txt 20 -e \
           -v 200 > output/example_1_2_share
```

8 Post-processing using R

polyDFE is accompanied by a series for R functions that allow

- parsing the output of polyDFE for easy manipulation in R
- summarizing the DFE estimated by polyDFE
- calculating α using alternative definitions
- bootstrapping data
- performing model testing
- performing model averaging
- creating init_file containing the best found parameters
- creating grouping for the r parameters

8.1 Parsing the output of polyDFE

The function

```
parseOutput(filename)
```

allows the user to parse the output of polyDFE from multiple runs stored in filename. The function returns a list with entries for each separate run of polyDFE found, containing

- the name of the input file used;
- the DFE model used;
- the best likelihood found and the corresponding gradient;
- the values of all parameters (including those that have been fixed using the flag 1 in init_file, see -i for details);
- which parameters have been estimated;
- the value of n (the sample size of the data);
- expected SFS and, if relevant, divergence counts and misattributed polymorphism;
- estimates of α.

8.2 Summarizing the DFE

The function

```
getDiscretizedDFE(estimates, sRanges = c(-100, -10, -1, 0, 1, 10))
```

discretizes the DFE found in estimates, which is one entry from the list returned by parseOutput. The discrete categories are given as a vector in sRanges.

8.3 Calculating α

The function

```
estimateAlpha(estimates, supLimit = 0, div = NULL, poly = TRUE)
```

calculates both α_{div} (when `div = NULL`) and α_{dfe} (when `div` is given) (Tataru et al., 2017), using the parameters found in `estimates`, which is one entry from the list returned by `parseOutput`.

Galtier (2016) argues that mutations with positive selection coefficients that are not higher than a certain threshold should, effectively, be treated as neutral mutations when calculating α (see also Tataru et al. (2017)). This threshold can be controlled here by setting `supLimit`.

When used, `div` should contain divergence data, which can be read using the function

```
parseDivergenceData(filename)
```

where `filename` is the same data file used for running `polyDFE`, see -d for details.

As described in Tataru et al. (2017), divergence data can contain misatributed polymorphism. This is the default behavior, but the correction can be turned off by setting `poly = FALSE`.

8.4 Bootstrapping data

The function

```
bootstrapData(inputfile, outputfile = NULL, rep = 1)
```

creates bootstrap data from the data found in `inputfile`, which is the same data file used for running `polyDFE`, see -d for details. The output is written to files with names `<outputfile>_i` for $1 \leq i \leq$ `rep`. By default, `outputfile = NULL`, and the function writes to `<inputfile>_i` instead.

8.5 Model testing

The function

```
compareModels(est1, est2 = NULL, nested = NULL)
```

compares the models found in `est1` and `est2` by calculating the AIC and, where relevant, the LRT.

Both `est1` and `est2` can either be lists returned by `parseOutput`, or file names, as for `parseOutput`. If these contain more then one run of `polyDFE`, then run i from `est1` is compared to run i from `est2`.

The function can automatically detect if models are nested for calculating the LRT, however nestedness can be enforced by setting `nested = TRUE`.

The function returns a list containing two entries

- `AIC` is a matrix containing the number of estimated parameters, the log likelihood and AIC for `est1` in columns $1-3$ and for `est2` in columns $4-6$, for all runs of `polyDFE` found in each row;
- `LRT` is a matrix containing the degrees of freedom in column 1, the likelihood of `est1` in column 2, the likelihood of `est2` in column 3 and the p-value from the LRT test in column 4, for all runs of `polyDFE` found in each row.

If `est2 = NULL`, only the AIC is calculated for `est1`.

8.6 Model averaging

The function

```
getAICweights(estimates)
```

calculates AIC weights (Posada and Buckley, 2004) for the runs of `polyDFE` found in `estimates`, a list as returned by `parseOutput`. The weights can the be used to calculate model avearge for a parameter of interest, such as any DFE parameter, or a discretized DFE (see `getDiscretizedDFE`)), α, or any other calculable quantities.

8.7 Creating `init_file`

The function

```
createInitLines(estimates, outputfile, startingID = 1,
                fix = c("eps_cont"), share = "all", groupingDiff = NA)
```

writes the values of the parameters found in `estimates`, a list as returned by `parseOutput`, or a file name, as for `parseOutput`. For each run of `polyDFE`, a new line is appended to `<outputfile>_init` containing the values of the best parameters found in the format described previously (see `-i` for details).

The `ID`s for each line are consecutive and start at `startingID`.

The parameters given in `fix` are flagged with 1, the ones in `share` are flagged with 2, while the rest of the parameters are flagged with 0. If `fix = "all"`, than all parameters are flagged with 1, and, similarly, if `share = "all"`, than all parameters are flagged with 2.

Grouping of r parameters (see `-g` for details) can be calculated automatically by setting `groupingDiff`. If the difference between r_i and r_{i+1} is smaller than `groupingDiff`, than they are placed in the same group. The grouping information is appended to `<outputfile>_grouping`, with `ID` matching to the `ID` in `<outputfile>_init`.

8.8 Examples

The accompanying file `example.R` shows how to post-process the `polyDFE` output files generated in 7.4.6.

Index

References

N. Galtier. Adaptive protein evolution in animals and the effective population size hypothesis. *PLoS genetics*, 12(1), 2016.

E. Jones, T. Oliphant, P. Peterson, et al. SciPy: Open source scientific tools for Python, 2001–. URL `http://www.scipy.org/`. [Online; accessed 2016-09-21].

D. Posada and T. R. Buckley. Model selection and model averaging in phylogenetics: advantages of akaike information criterion and bayesian approaches over likelihood ratio tests. *Systematic biology*, 53(5):793–808, 2004.

E. O. Purisima and H. A. Scheraga. An approach to the multiple-minima problem in protein folding by relaxing dimensionality: Tests on enkephalin. *Journal of Molecular Biology*, 196(3):697–709, 1987.

P. Tataru and T. Bataillon. polyDFEv2.0: Testing for invariance of the distribution of fitness effects within and across species. *bioRxiv*, 2018. doi: https://doi.org/10.1101/363887.

P. Tataru, M. Mollion, S. Glémin, and T. Bataillon. Inference of distribution of fitness effects and proportion of adaptive substitutions from polymorphism data. *Genetics*, 207(3):1103–1119, 2017.

D. J. Wales and J. P. Doye. Global optimization by basin-hopping and the lowest energy structures of Lennard-Jones clusters containing up to 110 atoms. *The Journal of Physical Chemistry A*, 101(28):5111–5116, 1997.

D. J. Wales and H. A. Scheraga. Global optimization of clusters, crystals, and biomolecules. *Science*, 285(5432):1368–1372, 1999.