RIDGE User manual V1.0

Burban Ewen

February 13, 2024

Contents

1 User manual **1**
1.1 RIDGE v1 . . . 2
1.2 Installation . . . 2
1.2.1 Requirements . . . 2
1.2.2 Get the code . . . 2
1.2.3 Install containers . . . 2
1.2.4 Set-up config folder . . . 2
1.3 Input file . . . 3
1.3.1 Vcf file . . . 3
1.3.2 Config.yaml file . . . 3
1.3.3 Popfile . . . 5
1.3.4 Contig data file . . . 5
1.3.5 Recombination rate data . . . 6
1.4 Usage . . . 6
1.4.1 Gather all necessary information . . . 6
1.4.2 Scan launch . . . 6
1.4.3 Complete launch . . . 6
1.5 RIDGE Pipeline . . . 6
1.5.1 Genome splitting and scanning . . . 8
1.5.2 Locus data gathering, subsampling and average summary statistics . . . 8
1.5.3 Generating priors and simulated datasets . . . 8
1.5.4 Inferring average demographic history . . . 8
1.5.5 Estimating the goodness of fit of prior and posterior parameters . . . 9
1.5.6 Simulating locus reference table and Detecting barrier . . . 9
1.6 Example of usage of RIDGE and recommendations . . . 9
1.6.1 Setup . . . 9
1.6.2 Scan genome & setup prior bounds . . . 10
1.6.3 Test prior bounds . . . 10
1.6.4 Analyze outputs . . . 11

1 User manual

This manual is intended to guide users through the installation and use of RIDGE by describing each script function and the resulting files in detail. The following notations help clarifying the nature of the different elements in the pipeline in this section :

- `Gray highlighted` text stands for input parameters of RIDGE that are used in config.yaml file (see Input files section).
- ***Italic bold*** text stands for files or programs needed in RIDGE.
- ***`Italic bold highlighted`*** text stands for RIDGE scripts files.

1.1 RIDGE v1

RIDGE takes as input a vcf file containing sequences of individuals from two populations, accompanied by accessory files providing complementary information. From this, RIDGE first uses Approximate Bayesian Computation (ABC) to infer demographic data by simulating 14 demographic x genomic models to produce a reference table. This table serves to train a random forest (RF) that generates weights and parameter estimates for each model according to their fit to the target (observed) dataset. Second, RIDGE constructs a hypermodel where the posterior distribution of each parameter is obtained as the weighted average over the 14 models. Finally, it uses this hypermodel to simulate one set of control loci (thereafter non-barrier) and one set of barrier loci that have undergone no gene flow during divergence. Simulated datasets generated for barrier and non-barrier loci are used to train a second RF that generates posterior probabilities and associated Bayes factors for each locus to belong to the barrier or non-barrier category.

1.2 Installation

1.2.1 Requirements

RIDGE uses the Snakemake workflow management system as well as Singularity containers. First ensure that Singularity and Snakemake (v 7.7.0) are installed on the machine where RIDGE will run. If it's not the case, contact the admin system in case of a cluster installation; otherwise, follow the installation instructions: `https://snakemake.readthedocs.io/en/stable/getting_started/installation.html` (pip installation is recommended)

1.2.2 Get the code

Download the code (v1.0, which is the version use in Burban et al. (2023)) with the following command

```
git clone -b v1.0 https://github.com/EwenBurban/RIDGE.git
cd RIDGE
```

1.2.3 Install containers

After completing this step, you will be provided with a list of .sif files in the container folder, including python.sif, R.sif, R_visual.sif and scrm_py.sif.

cluster installation To install RIDGE on a cluster, create a free account on https://sylabs.io and then go to https://cloud.sylabs.io/tokens to create an authentication token. Afterwards, input the command below and paste your token.

```
singularity remote login
```

Go into the RIDGE folder and launch container creation using the following command.

```
cd <path to RIDGE> bash cluster_configure.sh
```

This process installs all required programs and their dependencies within the container folder.

local installation If you install RIDGE on a local machine, simply execute the following command:

```
bash configure.sh
```

1.2.4 Set-up config folder

The configuration folder must contain at least one file (***config.sh***) that can tailor the behavior of RIDGE to fit your specific installation. As it is not included in the git clone, you will need to create the folder first. The content of this folder is called as the launch of RIDGE by ***RIDGE.sh***.

```
cd <path to RIDGE>
mkdir config
cd config
touch launch_param.sh
```

launch_param.sh This file regulates the number of jobs that snakemake will attempt to execute simultaneously and monitors snakemake. So open and then edit it using the following instruction and example:

Example of file :

```
module load snakemake singularity
mode='cluster'
ntask_load=140
ID=ridge_project
```

- The beginning of the file involves calling Snakemake and Singularity, which is essential in case these programs are not available by default in your working environment – often the case while using clusters. The command to call them is highly dependent on your installation, so do not take into account the command used in the example file. If you place Snakemake and Singularity within a Conda environment, this is the location to invoke the Conda environment 'conda activate <name of your env>.
- **mode** option defines the behavior of Snakemake. If the mode is set to 'cluster', then RIDGE will initiate jobs using SLURM. Afterwards, the cluster.json file must be completed (further information regarding this file is provided below). For 'local' mode, Snakemake will start jobs automatically. This mode is recommended for local installations or clusters that do not employ SLURM job managers.
- **ntask_load** option allows you to define the number of jobs that Snakemake will try to launch.
- **ID** option is specific to mode='cluster', as it defines the name of the user who starts the job. This only applies to the SLURM structure, and is commonly your user name. If mode is set to 'local', this line can be removed.

cluster.json This document outlines the resources available to each job and enables you to fine-tune and optimize RIDGE for your specific cluster. Further information regarding cluster configuration can be found in the cluster configuration section of `https://snakemake.readthedocs.io/en/stable/snakefiles/configuration.html`. An example is also available at template/cluster.json. Input files To launch RIDGE, you need to provide at least four files in your work folder (i.e the folder where RIDGE will work and generate output). So, your work folder must follow the following configuration before any launch:

work_dir/
├─ vcf_file
├─ contig_data.txt
├─ popfile.csv
├─ config.yaml
├─ rec_rate_map (optional

1.3 Input file

1.3.1 Vcf file

In the actual version, RIDGE only takes as genomic polymorphism data a vcf file (vcf file format >= 4.0). RIDGE can manage haploid and diploid data. The vcf file must contain only biallelic sites. See `https://en.wikipedia.org/wiki/Variant_Call_Format` for detailed information on the file format

1.3.2 Config.yaml file

The config file contains all the data to start RIDGE. Note, that in this file, the priors used in the ABC process are defined, and so, an incorrect specification of the hyperpriors can drastically affect the performances and results of RIDGE The fields of the config file are the followings:

- `config_yaml`: the name of the ***config.yaml*** file that you are actually filling. (Note that you do not need to give the absolute path, but only the filename, otherwise it will stop)
- `vcf_file`: the name of vcf file (only filename expected)
- `contig_data`: the name of the contig data file (only filename expected)

- `rec_rate_map`: the name of the recombination map (only filename expected or NA if no map)
- `popfile`: the name of the popfile (only filename expected)
- `nameA` and `nameB`: name of one of the two populations. The names must be the same as used in popfile
- `container_path`: the absolute path to the container folder, which contain all the Singularity container, and so all programs
- `ploidy`: the level of ploidy of the dataset. 1 is for haploid, 2 is for diploid
- `ligthMode`: Activate the ligthMode, which is a fastest but less precise version of RIDGE
- `work_dir`: absolute path to the work folder
- `nLoci`: number of loci sampled, used to avoid unnecessary computational time. If nLoci is set to -1, all the genome will be used in the process (it may slow down the process by 10 to 100 times, depending on the size of the dataset). Note that a total number of loci around 1000 loci is a good trade-off between genome representation and computation time limitations
- `window_size`: size of each locus in bp Choose the value according to the SNP density in your data.
- `homo_rec`: If True, recombination rate is considered homogeneous along the genome, if False,it uses the recombination map provided with `rec_rate_map`
- `homo_rec_rate`: recombination rate value along the genome (used only if `homo_rec`=True)
- `mu`: mutation rate (assumed constant along the genome) per pb. Note that mu is needed for prior bound suggestion even if hetero_theta=True.
- `Nref`: Population size of reference (in number of individuals) used to rescale all values in coalescent units.
- `N_min` & `N_max`: minimum and maximum population size (in number of individuals). It is highly recommended to set the value on the basis of the real diversity value in the vcf rather than expected value from the literature (to have a good estimation you can launch RIDGE in scan mode and follow suggestion from ***prior_bound_suggestion.txt***).
- `M_min` & `M_max`: minimum and maximum migration rate (in $4 * Nref * m$ unit). By default, `M_min`=0.1 and `M_max`=50
- `Tsplit_min` & `Tsplit_max`: minimum and maximum time of split of the ancestral population in generations !!! Note that it is highly recommended to set the value on the basis of the real data in the vcf rather than expected value from the literature (to have a good estimation you can launch RIDGE in scan mode and follow suggestions from ***prior_bound_suggestion.txt***).
- `Pbarrier_max`: maximum proportion of the genome under barrier to gene flow. By default, `Pbarrier_max`=0.2 (i.e. 20% of loci are considered as barriers).
- `hetero_theta`: Activate/Deactivate (True/False) hetero θ option. If True, RIDGE ignores `mu` to set an expected level of diversity and rather uses θ computed on the observed number of SNPs using Watterson θ estimator: $\theta_W = S/a$.

Example of file:

```
M_max: 50
M_min: 0.1
N_max: 200000
N_min: 10000
Nref: 50000
Tsplit_max: 20000
Tsplit_min: 1000
Pbarrier_max: 0.2
config_yaml: config.yaml
```

```
container_path: /home/RIDGE/container
contig_data: contig_data.txt
lightMode: False
mu: 1e-8
nameA: wild
nameB: dom
popfile: popfile.csv
rec_rate_map: rho_map.txt
work_dir: /home/example_ridge
window_size: 50000
ploidy: 1
vcf_file: vcf_file.vcf
homo_rec: False
homo_rec_rate: NA
hetero_theta: True
nLoci: 1000
```

1.3.3 Popfile

This file lists the individuals from each population in csv format (with ',' as separator). The name of each population must be in the header. The popfile must contain at least two populations (so two columns) and each list must be of the same length as the others. If the populations are not of equal length, you can fill the missing individuals with NA.

Example of file:

```
wild,dom
W1,Dx1
W3,Dx10
W5,NA
W7,Dx11
```

1.3.4 Contig data file

A file containing the length of each chromosome/contig and their associated names and order.

- **contig_name**: is the name of the chromosome/contig in the vcf file
- **contig_length**: the length in bp of the chromosome/contig
- **index**: the index in the order of contigs

Example of file:

```
contig_name contig_length index
Chr1 43270923 1
Chr2 35937250 2
Chr3 36413819 3
Chr4 35502694 4
Chr5 29958434 5
Chr6 31248787 6
Chr7 29697621 7
Chr8 28443022 8
Chr9 23012720 9
```

1.3.5 Recombination rate data

RIDGE uses either a recombination map or a constant recombination rate to work. If you choose to use a constant recombination rate, you must set `homo_rec` to True in the ***config.yaml*** file and fill the field `homo_rec_rate` with the mean recombination rate estimated for your dataset. Otherwise, you need to fill the field `rec_rate_map` with the name of your recombination map and place it in the work folder. Note that the recombination rate r must be the number of recombination rate per bases and per generation and the recombination map uses the tabulation as a separator.

- **chr**: the index of the contig (see contig file)
- **start** and **stop**: the beginning and ending in bp of the window
- **r**: the recombination rate inside the window in bp

Example of file :

```
chr start end r
9 21800000.0 21900000.0 7.170443918444081e-07
9 21900000.0 22000000.0 6.771961140602021e-07
9 22000000.0 22100000.0 6.44356192372138e-07
9 22100000.0 22200000.0 6.08745943319314e-07
9 22200000.0 22300000.0 5.709059200375581e-07
```

1.4 Usage

1.4.1 Gather all necessary information

Before any launch you must fill and provide all mandatory input files. See section 1.3 for details.

1.4.2 Scan launch

To correctly fill your config files you will need a measure of the diversity and divergence. Under "scan" mode, you are not obliged to set up values for the prior bounds (`N_min, N_max, N_ref, Tsplit_min, Tsplit_max, M_min, M_max, Pbarrier_max`), but for "all" mode – which run RIDGE analysis – they are mandatory. You can choose to use your own prior bouds or to use on estimation done by RIDGE. If you choose to use your own prior, see section 1.6.3 to validate the quality of you prior, otherwise read the following part to generate prior suggestion. For `M_min ; M_max` and `Pbarrier` default values are suggested (`M_min`=0.1, `M_max`=50 and `Pbarrier_max`=0.2). Next you have to launch RIDGE in scan mode with the following command

```
bash <path to RIDGE>/RIDGE.sh <work_dir>/config.yaml scan
```

RIDGE will generate a file called ***prior_ bound_ suggestion.txt***. It is highly recommended to define prior bounds based on this file, as the values are calculated from the raw data. The procedure for validating prior bounds is explained in section 1.6.3.

1.4.3 Complete launch

Once `N_min, N_max, N_ref, Tsplit_min, Tsplit_max, M_min, M_max, Pbarrier_max` are correctly set, relaunch RIDGE, but this time with the whole process, without forgetting to delete ***modelComp/*** folder:

```
rm -r <work_dir>/modelComp <work_dir>/gof_prior.txt <work_dir>/QC_plot
bash <path to RIDGE>/RIDGE.sh <work_dir>/config.yaml all
```

1.5 RIDGE Pipeline

RIDGE relies on ABC for detecting gene flow barriers by incorporating the effect of demography. Firstly, RIDGE infers the parameters of a hyper demographic x genomic model; secondly, it infers the probability of each locus to be a barrier. Each step of the pipeline is detailed below in the order of execution and represented in Fig 1.

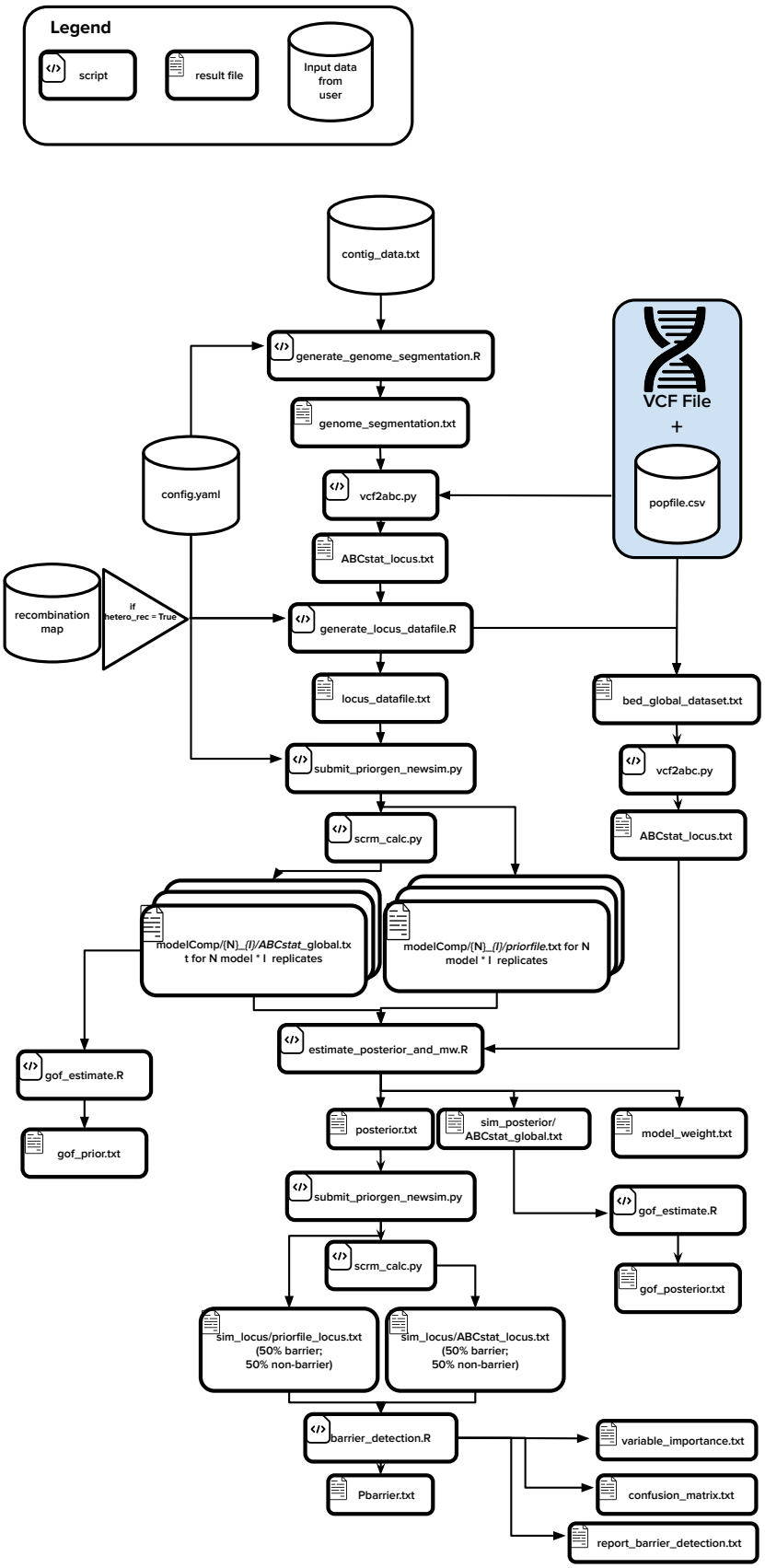

Figure 1: Graphical representation of RIDGE pipeline

1.5.1 Genome splitting and scanning

Initially, RIDGE takes the chromosome size information from the ***contig_ data.txt*** file and combines it with the `window_size` parameter provided in ***config.yaml***. The script ***generate_genome_segmentation.R*** then divides the genome into non-overlapping windows of `window_size` base pairs. The genome segmentation is stored in the file
genome_ segmentation.txt. This file is then used by ***vcf2abc.py*** to determine the boundaries of each locus, accompanied by ***popfile.csv*** which indicates the sample composition of each population, to calculate summary statistics on the vcf file. The computation method used for summary statistics is given in Burban et al. (2023). The summary statistics for each locus are stored in the file ***ABCstat_ locus.txt***.

1.5.2 Locus data gathering, subsampling and average summary statistics

The script ***generate_locus_datafile.R*** creates the ***locus_ datafile*** file containing the data needed to simulate loci in further steps. The data needed to simulate a locus are:

- The number of haploid samples from each population, for which we multiply the number of samples from the population by the ploidy level. To get this information, ***generate_locus_datafile.R*** uses ***popfile.csv*** combined with population names (`nameA` and `nameB`) and `ploidy` level from the ***config.yaml*** file.
- The total number of haploid samples, which is the sum of both population sizes multiplied by their `ploidy` level.
- The recombination rate within the locus is calculated as follows: $\rho = 4 * Nref * r * L$, where `Nref` is the reference population size, L is the `window_size` defined in ***config.yaml***, and r is the expected number of recombinations on the locus per generation and per individual. There are two ways to provide r to RIDGE. The first is to use a fixed value across the genome by setting `homo_rec` to 'True' and specifying the value of r through the `homo_rec_rate` option. The second is to provide a recombination map by setting `homo_rec` to 'False' and declaring the path file to the recombination map in `rec_rate_map`.
- θ represents the expected diversity. RIDGE offers two ways to compute it: 1) Using a fixed mutation rate across the genome (μ), by setting `hetero_theta` to 'False' and specifying the `mu` value in the ***config.yaml*** file and computing $\theta = 4 * Nref * \mu * L$. 2) Obtaining it from the observed amount of SNPs using $\theta_W = S/a$ (see Burban et al. (2023) for details on measuring genomic diversity). This option allows for the consideration of heterogeneity in mutation rates across the genome and/or variations in data coverage (refer to Burban et al. (2023) for the rationale behind this option).

For each locus, we obtain the necessary information for simulations. Then, we randomly select `nLoci` loci across the genome without replacement. This reduces computation time and avoids unnecessary simulations for demographic model inference, which accounts for 80% of the RIDGE running time. For optimal results, we recommend using `nLoci`=1000. Locus data is stored in the ***locus_ datafile***, and the genomic boundaries of the `nLoci` loci are stored in ***bed_ global_ dataset.txt***. The program ***vcf2abc.py*** uses ***bed_ global_ dataset.txt***, ***vcf file***, and ***popfile.csv*** to compute the average summary statistics across `nLoci` loci stored in ***ABCstat_ global.txt*** (see Burban et al. (2023) for summary statistics computation method).

1.5.3 Generating priors and simulated datasets

The script ***submit_priorgen_newsim.py*** generates prior simulation parameters using the method explained in Burban et al. (2023). The parameters are stored in
modelComp/{N}_ {I}/priorfile.txt (where N is the model name and I is the number of replicates) and in a *scrm* compatible command format. After generating the prior parameters, simulations are run from them using *scrm* (Staab et al. 2015). The results of these simulations are then piped directly into ***scrm_calc.py***, which transforms the simulation results into summary statistics using the same method explained in Burban et al. (2023). The results of ***scrm_calc.py*** are stored in ***modelComp/{N}_ {I}/ABCstat_ global.txt***.

1.5.4 Inferring average demographic history

The modelComp folder contains the ?reference table? ? a table containing the parameter for simulations and the summary statistics ? on which a RF is trained to infer each parameter. Each model among the 14 demographic x genomic models, has a different number of parameters. So at first ***estimate_posterior_and_mw.R***, put each

model under the ?hypermodel? ? a model that uses all parameters, all models combined ? by filling missing parameters (see Table S1 in Burban et al. (2023)). Then, for each of the 12 parameters of the hypermodel, a regression RF (*regAbcrf* from *abcrf* R packages Raynal et al. (2019)) is trained. It predicts from summary statistics of the observed dataset (stored in ***ABCstat_global.txt***), the parameters values. Joint posteriors and model weights are generated following the procedure explained in Burban et al. (2023) and stored in ***posterior.txt*** and ***model_weight.txt***. The summary statistics of ***posterior.txt*** are stored in ***sim_posterior/ABCstat_global.txt***.

1.5.5 Estimating the goodness of fit of prior and posterior parameters

The goodness of fit of the posterior distributions (from ***sim_posterior/ABCstat_global.txt***) and prior (from ***modelComp/{N}_{I}/ABCstat_global.txt***) are evaluated using an enhanced version of the *gfit* function of the abc packages (Csillery et al. 2012) coded in ***gof_estimate.R***, which uses a goodness-of-fit statistical approach described in Burban et al. (2023). The results of the posterior and prior goodness of fit are stored in ***gof_posterior.txt*** and ***gof_prior.txt***, respectively.

1.5.6 Simulating locus reference table and Detecting barrier

The ***posterior.txt*** file, containing posterior parameters, undergoes transformation into simulation parameters, with half of simulation migration set to $m = 0$ and the second half $m > 0$ using the ***submit_priorgen_locus.py*** script and is subsequently stored in ***sim_locus/priorfile_locus.txt***. In contrast to the previous simulation step, where only average information across `nLoci` were retained, this iteration preserves information at the locus scale. Similar to the ***submit_priorgen_newsim.py*** procedure, simulation parameters are formatted in *scrm* command style and executed using *scrm*. The resulting simulations are then immediately transformed into summary statistics through the utilization of ***scrm_calc.py***. This time, summary statistics at the locus level are stored in ***sim_locus/ABCstat_locus.txt***. Combining ***sim_locus/priorfile_locus.txt*** and ***sim_locus/ABCstat_locus.txt*** forms the "locus scale reference table", on which the RF algorithm *abcrf* (from the *abcrf* package Raynal et al. (2019)) is trained to classify barriers ($m = 0$) and non-barriers ($m > 0$). Subsequently, using summary statistics for each locus from ***ABCstat_locus.txt***, the RF classifies each locus into the barrier or non-barrier class. From the RF outcomes, Bayes factors are computed (refer to Burban et al. (2023) for details on bayes factor computation). The Bayes factors and corresponding summary statistics for each locus are stored in ***Pbarrier.txt***. Variable importance for each variable, following the Random Forest model, is stored in ***variable_importance_barrier.txt***. Additionally, the confusion matrix, as defined in the Random Forest context, is stored in ***confusion_matrix_barrier.txt***. Finally, the average barrier proportion and resulting barrier/non-barrier ratios are stored in ***barrier_proportion_and_ratio.txt***.

1.6 Example of usage of RIDGE and recommendations

The RIDGE code comes with a small test dataset in the ***example/*** folder. In this part, I describe step-by-step the process of using RIDGE on a dataset and how to interpret RIDGE's outputs. The dataset is a subset of the published dataset in Poelstra et al. (2014) and Vijay et al. (2016), stored at NCBI under PRJNA192205, containing 9 out of 1300 scaffolds, where SNPs were called on the following reference genome GCF_000738735.1 (available at NCBI). This subsample focuses on the 9 scaffolds where genes of interest were found in Poelstra et al. (2014). The individuals used are the same as in Poelstra et al. (2014). The example folder contains :

- An archive name test_dataset.vcf.tar.gz
- A preset contig_data.txt
- A preset popfile.csv
- A list of the genes of interest detected in Poelstra et al. (2014) and their positions

1.6.1 Setup

Prepare files : First, go in the ***example*** folder and decompress the vcf file:

```
cd example
tar -xvzf test_dataset.vcf.tar.gz
```

Then, create the file ***config.yaml***, by taking the template from template folder:

parameter	"Good" prior bounds	"Bad" prior bounds
N_min	42500	140500
N_max	147500	300500
Tsplit_min	5000	100000
Tsplit_max	120000	120000
M_min	0.1	0.1
M_max	50	50
Pbarrier_max	0.2	0.2

Table 1: Values of prior bound for "good" (generated based on ***prior_bound_suggestion.txt***) and "bad" (value choosen arbitrary) prior bounds

```
cp ../template/RIDGE_template.yaml config.yaml
```

Adapt the content from config.yaml to your installation (follow instruction from 1.3). To correctly fill your config files you will need a measure of the diversity and divergence. At first, you are not obliged to set up values for the prior bounds (`N_min, N_max, N_ref, Tsplit_min, Tsplit_max, M_min, M_max, Pbarrier_max`), but in the end they are mandatory. Here, leave all parameters empty exept for `M_min, M_max` and `Pbarrier` to fill with the following value : `M_min`=0.1, `M_max`=50 and `Pbarrier_max`=0.2.

1.6.2 Scan genome & setup prior bounds

Launch RIDGE in scan mode,

```
bash <path to RIDGE>/RIDGE.sh <work_dir>/config.yaml scan
```

The genome will be scanned and summary statistics will be generated for each window. RIDGE also suggest prior bounds based on statistics (available in
prior_bound_suggestion.txt). It is advisable to utilize these values as they are scaled to the `mu` value provided in the config.yaml file. In the event of any changes to the `mu` values, rerun or re-adjust other values.

1.6.3 Test prior bounds

!!! Inference power of RIDGE depends on the quality of priors. So, pay attention to prior bounds. In the following steps, it explains how to measure the quality of prior bounds !!!

To test prior bounds, launch RIDGE in test mode using the following command:

```
bash <path to RIDGE>/RIDGE.sh <work_dir>/config.yaml test
```

This way, only 1% of the simulation in ***modelComp*** folder are launched, then the goodness of fit of prior is evaluated in ***gof_prior.txt*** and ***QC_plot/QC_prior_acp.pdf*** and ***QC_plot/QC_prior_density.pdf***. To demonstrate the significance of selecting a suitable prior bound, I executed the RIDGE in test mode with a "good" and a "bad" choice of priors. For the "good" choice, I used the value suggested in the ***prior_bound_suggestion.txt*** file, while for the "bad" choice, I chose a version biased towards higher values.

In the "bad" prior bound choice, the observed values fall outside the distribution range for piA_avg, piB_avg, and divAB_avg statistics (respectively the π of population A and B and the D_{xy} between both populations). This shows that the prior bounds do not include the true observed value, contrary to "good" prior bound choice (see Figure 2). It is necessary to evaluate the quality of prior bounds by examining the prior density in
QC_plot/QC_prior_density.pdf before running the entire RIDGE. Additionally, users may evaluate the quality through PCA on summary statistics (accessible at
QC_plot/QC_prior_acp.pdf), but this method is less sensitive and less informative. With this visual approach, the prior must include the observations to validate the prior. Finally, the goodness of fit (GOF) test can be used, but it is only reliable if the threshold is set at 5%, indicating a "bad" prior estimation. If the observations demonstrate a poor quality of prior bounds, follow these steps:

```
cd <work_dir>
rm -r modelComp/ QC_plot/ gof_prior.txt
```

And then test relaunch RIDGE until you achieve satisfactory prior bounds. It is recommended to use large prior bounds as there is little cost associated with doing so. !!! For N and Tsplit, avoid having more than two orders of magnitude between your min and max bounds. !!!

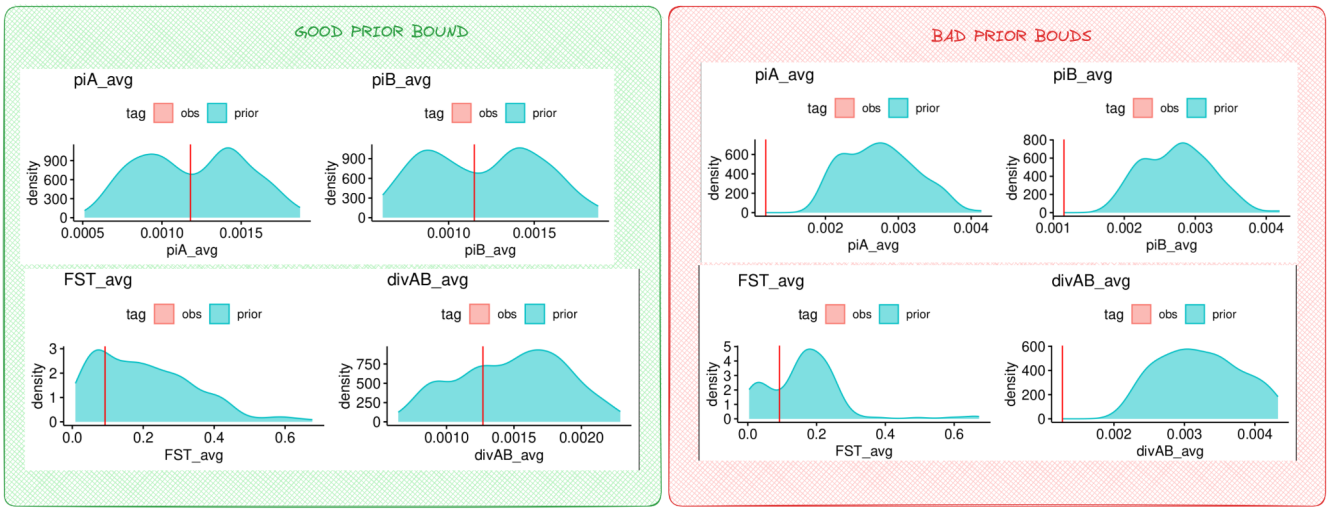

Figure 2: Distribution of Summary Statistics from Simulation Generated Within Prior Bounds. "Obs" represents the observed dataset value for the summary statistic, while "Prior" denotes the distribution of summary statistics generated from simulations within the specified prior bounds. The "Good" choice of prior is deemed favorable, as the observed value is contained within 95% CI values of the prior distribution. Conversely, the "Bad" choice shows observed values falling in the distribution tail or outside the bounds of the prior distribution.

1.6.4 Analyze outputs

To launch RIDGE use the following command :

```
bash <path to RIDGE>/RIDGE.sh <work_dir>/config.yaml all
```

Depending on the `lightMode` option, the runtime is around 470000 s for `lightMode`=True and 1883000 s for `lightMode`=False (assuming that you run RIDGE on 4 cores running at least at 2.5GHz) for a dataset with a ρ/θ=20 with `nLoci`=1000 and `window_size`. Run-time can be multiply by 3-4 for high ρ/θ (e.g.ρ/θ=500).

Check the quality of demographic infereneces

To evaluate the quality of demographic inferences, there are two levels of verification:

- Agnostic Level: At this level, the assessment involves confirming that the posterior aligns more closely with the observed dataset than the prior.
- Documented Level: This level involves checking whether the parameter values are consistent with established knowledge. The second level is optional and relies on the user's familiarity with the dataset.

For the first level of checking, we use visual observation of posterior distributions (available in ***QC_plot/QC_posterior_de*** and ***QC_plot/QC_posterior_acp.pdf***) and evolution of goodness of fit between ***gof_prior.txt*** and ***gof_posterior.txt***. We expect posterior distributions to be more centered on the obs dataset and the distribution less wide than for priors (as observable Fig 3). Furthermore, the goodness of fit should increase between prior and posterior. In our example the goodness of fit slightly increased from 0.26 (for prior) to 0.31 (for posterior).

The second level of verification relies on pre-existing knowledge. For this dataset, T_{split} is anticipated to be approximately 80,000 generations (Poelstra et al. 2014; Vijay et al. 2016), with an effective population size ranging between 100,000 and 300,000 individuals. The demographic model is expected to align with a secondary contact model. Results, available at ***model_weight.txt*** and ***visual_model_weight.pdf***, indicate a predominant contribution of the IM_2M_2N and SC_2M_2N models to the estimation of demographic history, which is consistent with existing data. Indeed, both models (IM & SC) involve ongoing migration and exhibit heterogeneity in both migration (2M) and effective population size (2N)(see Figure 4). The estimated value (available at ***posterior.txt*** and ***visual_posterior.pdf***) of T_{split} closely matches the expected value (average T_{split} = 73414 generations), and the population size falls within the anticipated interval ($\approx 100,000$ for N1, N2, and Na)(see Figure 5). The migration rate is higher for current migration (M_{cur}) than ancestral migration (M_{anc}), suggesting an increase in migration over time, even if the estimated model is not optimal.

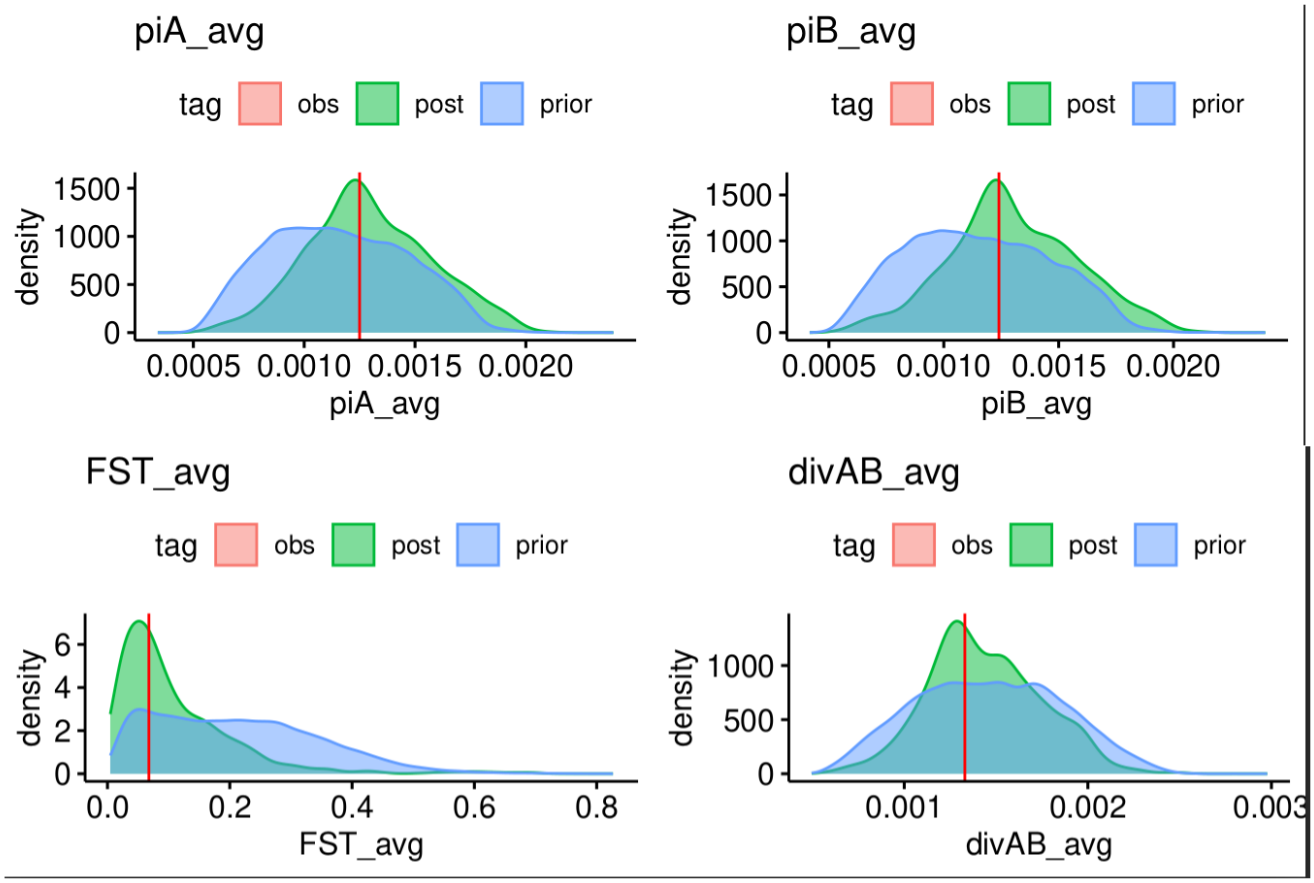

Figure 3: Distribution of Summary Statistics for Prior and Posterior Simulations. "Obs" represents the observed dataset value for the summary statistic, "Prior" denotes the distribution of summary statistics generated from simulations based on the prior, and "Post" illustrates the distribution of summary statistics from simulations of the posterior.

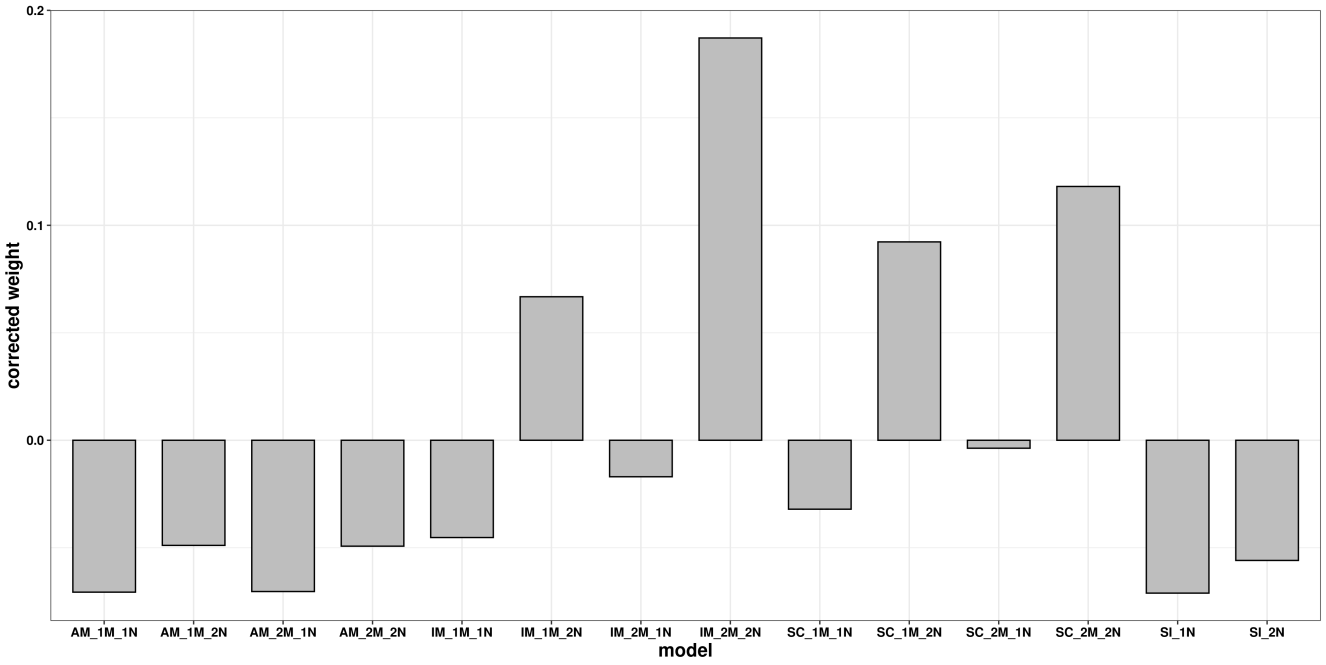

Figure 4: Model weight of each 14 model corrected by the uniform distribution model weight

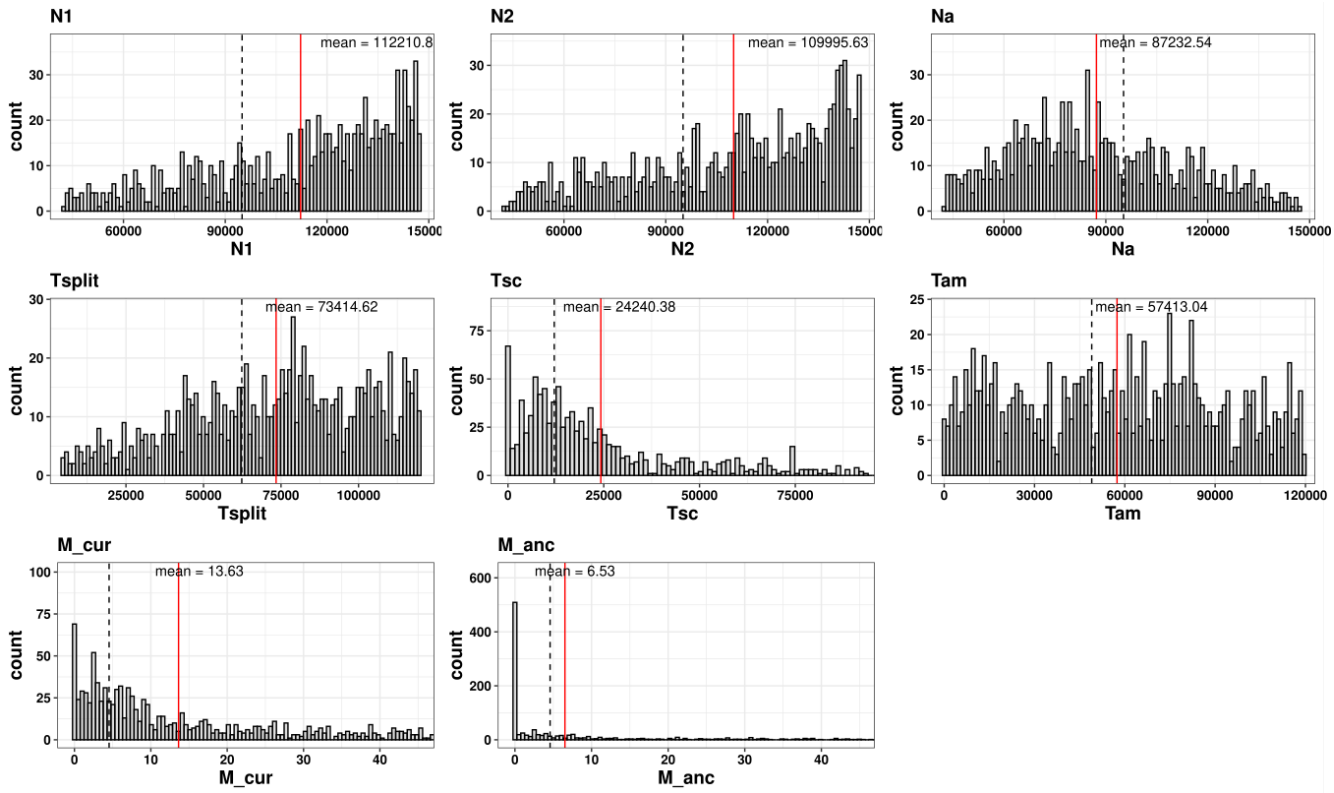

Figure 5: Distribution of parameter posterior values for N_1, N_2, N_a, T_{split}, T_{SC}, T_{AM}, M_{cur}, and M_{anc}. Dashed lines represent the mean value of priors, and the red line represents the mean value of posterior.

Allocation\true status	barrier	non-barrier	class.error
barrier	95695	1464	0.0150
non-barrier	1285	101407	0.0125

Table 2: Confusion matrix produced for the example case

Detecting barrier to gene flow

In this section, I distinguish between 1) detecting barriers to gene flow for a specific dataset without a comparative framework and 2) detecting barriers to gene flow across multiple datasets with the goal of comparing results between datasets. This difference is crucial as it affects how the user should utilize the different outputs.

Quality of barrier detection

Test of barrier detection quality relies on two elements: i) the confusion matrix of the random forest and ii) the distribution of variable importance of the random forest. In our example, the obtained confusion matrix (available in ***confusion_matrix_barrier.txt***) shows the distribution of allocation of simulated loci used to train the random forest. It demonstrates the ability of the RF to accurately classify the simulated loci. Here, 95695 loci simulated as barriers have been classified as barrier and 1464 as non-barrier, corresponding to an error of 1.5% (see Table 2). The higher the class error, the lower the confidence in the results. A high degree of class error (class.error=0.5) indicates that the Random Forest (RF) is unable to distinguish between barrier and non-barrier loci.

Furthermore, the variable importance, available at ***variable_importance_barrier.txt***, provides insights into which variables enable the RF to distinguish between barrier and non-barrier classes. Variable importance is calculated by considering how much each feature contributes to differentiate barrier from non-barrier classes across all the trees. Higher importance values indicate more influential features. The expectation was that some summary statistics such as ss, F_{ST}, netDivAB ($=D_a$), and sf would be important for barrier barrier detection. Our results are consistant with this prediction (see Fig 6). A pattern where all variables have the same importance reduces confidence in the results, as the RF is then unable to prioritize specific summary statistics to detect barrier loci.

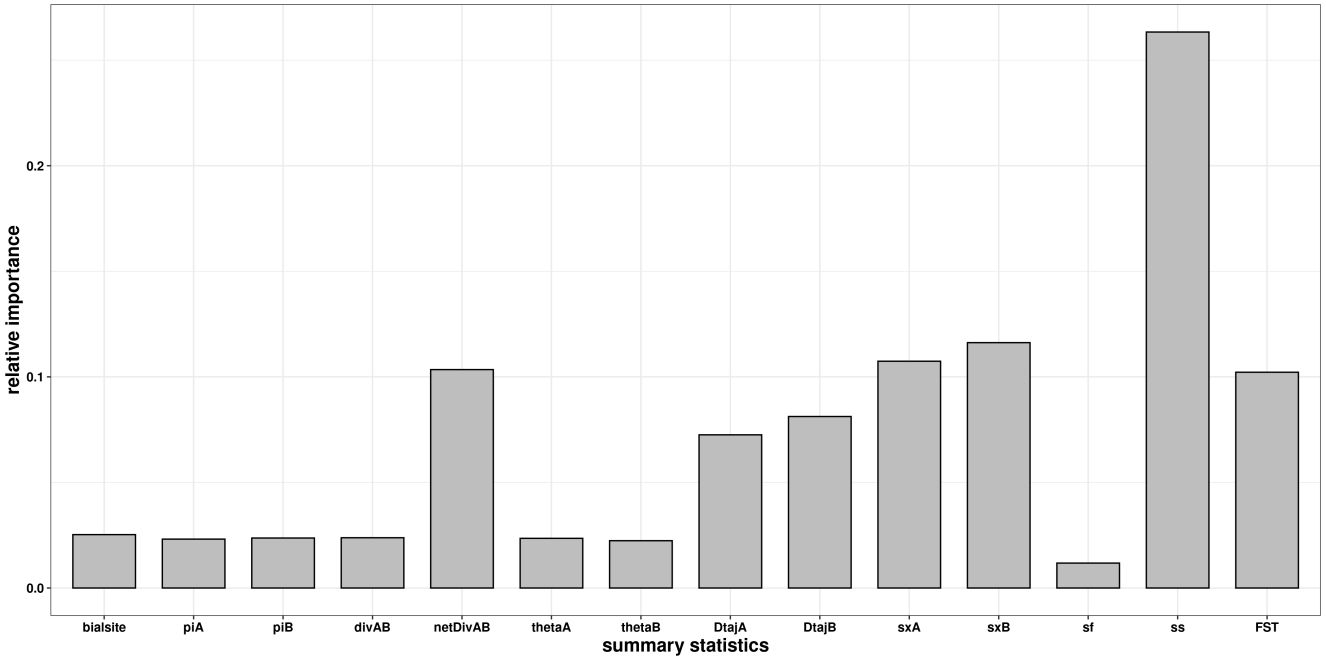

Figure 6: Relative importance associated with each summary statistic during the random forest building for the example case.

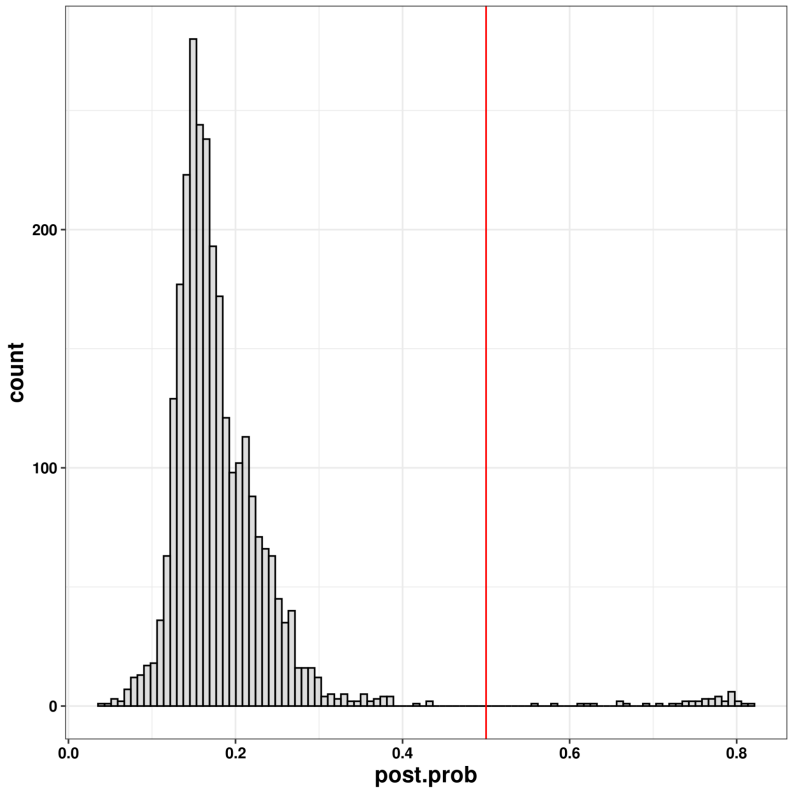

Figure 7: Distribution of posterior probability of barriers across the genome of the example data. The threshold for post.prob is set to 0.5 with a red line.

Detecting barrier for a focal dataset

RIDGE provides a posterior probability (post.prob) for each locus to quantify the probability of it being a barrier, which is accessible through ***Pbarrier.txt***. Loci with a post.prob > 0.5 have a high chance of being a barrier. By nature post.prob are not comparable from one dataset to another, as they are conditionated to demographic model. So, if there is only one dataset post.prob could be used. We suggest setting a threshold of at least $post.prob > 0.5$, and even stricter if necessary. In the example presented in Figure 7, a threshold of 0.5 effectively distinguishes the two observable groups of loci in the distribution.

Detecting barrier across multiple dataset

RIDGE provides, for each locus, a posterior probability and the Bayes Factor (BF) (refer to Burban et al. (2023) for details). By default, a BF is interpreted as follows: BF=10 means that the locus has 10 times more chance to be a barrier than a non-barrier, given Q. A BF=100 indicates a very high probability that the locus is a barrier. Compared to post.prob, BF is comparable from one dataset to another and allow the user to compare multiple datasets. To compare multiple datasets, the user must choose a threshold in BF value that is applied consistently across all datasets. A BF value of 100 is a relevant starting point, but it may not be suitable for all datasets. Indeed, depending on the datasets, there might be significant variations in the scales of BF (e.g., one dataset may range BF values between 0 and 100, while another ranges between 0 and 3000). This variability can occur when a dataset has a low T_{split}, causing BF to be biased toward high values. In such cases, we recommend using the "BF_approxQ" column in ***Pbarrier.txt***, as the approximation is more reliable under these circumstances (see Burban et al. (2023) for details).

References

Burban, Ewen, Maud Irene Tenaillon, and Sylvain Glemin (Sept. 17, 2023). *RIDGE, a tool tailored to detect gene flow barriers across species pairs*. Pages: 2023.09.16.558049 Section: New Results.

Csillery, Katalin, Olivier FranÃ§ois, and Michael G. B. Blum (2012). "abc: an R package for approximate Bayesian computation (ABC)." In: *Methods in Ecology and Evolution* 3.3, pp. 475–479.

Poelstra, J. W. et al. (June 20, 2014). "The genomic landscape underlying phenotypic integrity in the face of gene flow in crows." In: *Science* 344.6190, pp. 1410–1414.

Raynal, Louis et al. (May 15, 2019). "ABC random forests for Bayesian parameter inference." In: *Bioinformatics* 35.10, pp. 1720–1728.

Staab, Paul R. et al. (May 15, 2015). "scrm: efficiently simulating long sequences using the approximated coalescent with recombination." In: *Bioinformatics* 31.10, pp. 1680–1682.

Vijay, Nagarjun et al. (Oct. 31, 2016). "Evolution of heterogeneous genome differentiation across multiple contact zones in a crow species complex." In: *Nature Communications* 7.1. Number: 1 Publisher: Nature Publishing Group, p. 13195.