A tutorial on how to use BayeScan & BayeScenv

Running BayeScan

The method has been implemented in C++ and there are command-line versions for all operating systems (OS X, Lynux, and Windows). There is a GUI version only for Windows. Here we will use the command-line version.

Input File:

Genetic data:
The best approach is to prepare files with FSTAT, popgen, or Arlequin format. Then use PGDSpider
http://www.cmpg.unibe.ch/software/PGDSpider/
to convert them into GESTE/BayScan format.

Output files

There is a large number of files that can be obtained by running BayeScan. The most important ones, which are also the ones printed by default, are:

_Verif.txt: you should use this file in order to verify that bayescenv read the input files properly.

.sel: provide the trace of some of the parameters estimated by the model. This include the loglikelihood and the local population Fst's. This file is used to verify convergence of the chain.

_fst.txt: this file is used to interpret the results. In other words, identify the outlier loci.

Command line:

If you type "./BayeScan2.1_linux64bits", you get the list of options with a brief explanation

```
----------------------------
| BayeScan 2.0 usage:      |
----------------------------
-help         Prints this help
----------------------------
| Input                    |
----------------------------
alleles.txt  Name of the genotypes data input file
-d discarded Optional input file containing list of loci to discard
-snp         Use SNP genotypes matrix
----------------------------
| Output                   |
----------------------------
-od          Output file directory, default is the same as program file
```

```
-o alleles   Output file prefix, default is input file without the
             extension
-fstat       Only estimate F-stats (no selection)
-all_trace   Write out MCMC trace also for alpha paremeters (can be a very
             large file)
----------------------------
| Parameters of the chain |
----------------------------
-threads n   Number of threads used, default is number of cpu available
-n 5000      Number of outputted iterations, default is 5000
-thin 10     Thinning interval size, default is 10
-nbp 20      Number of pilot runs, default is 20
-pilot 5000  Length of pilot runs, default is 5000
-burn 50000  Burn-in length, default is 50000
----------------------------
| Parameters of the model |
----------------------------
-pr_odds 10  Prior odds for the neutral model, default is 10
-lb_fis 0    Lower bound for uniform prior on Fis (dominant data), default
             is 0
-hb_fis 1    Higher bound for uniform prior on Fis (dominant data),
             default is 1
-beta_fis    Optional beta prior for Fis (dominant data, m_fis and sd_fis
             need to be set)
-m_fis 0.05  Optional mean for beta prior on Fis (dominant data with -
             beta_fis)
-sd_fis 0.01 Optional std. deviation for beta prior on Fis (dominant data
             with -beta_fis)
-aflp_pc 0.1 Threshold for the recessive genotype as a fraction of maximum
             band intensity, default is 0.1
----------------------------
| Output files             |
----------------------------
-out_pilot   Optional output file for pilot runs
-out_freq    Optional output file for allele frequencies
```

The output files we will use were generated with the following command lines:

Long run that leads to results that are reliable because convergence of the MCMC was achieved. The prior odd is somewhat small (=10) for a dataset of 5000 SNPs so the number of false positives is likely to be large unless we use a very stringent FDR threshold.

```
./BayeScan2.1_macos64bits HsIMM-Us100-2.txt -o bayescanLongRunOD10 -
n 5000 -thin 10 -nbp 20 -pilot 5000 -burn 50000 -pr_odds 10
```

Long run (convergence of the MCMC was achieved). The prior odd is much larger than in the previous case and more appropriate for 5000 SNPs so the number of false positives will be smaller.

```
./BayeScan2.1_macos64bits HsIMM-Us100-2.txt -o bayescanLongRunOD10 -
n 5000 -thin 10 -nbp 20 -pilot 5000 -burn 50000 -pr_odds 100
```

A very important parameter of the method is the prior odds (PO) of the neutral model, highlighted in red in the above command lines. This parameter indicates our skepticism about the possibility that a given locus is under selection. In other words, it indicates how much more likely we think the neutral model is (which does not include the locus-specific effect α) compared to the model with selection (i.e. with the locus-specific effect α). For example a PO = 10 indicates that we think the neutral model is 10 times more likely than the model with selection. The larger the PO, the

more conservative the test of selection is. In principle, this parameter will not influence much the results if we have a large amount of data including many populations and many individuals per population. However, very frequently the number of populations or sample sizes are limited (e.g. less than 20 populations) so we do need to pay attention to this parameter. The PO value that should be used depends on how many loci are included in the data set. If there are less than 1000 loci, then PO = 10 is reasonable but with more loci (say between 1000 and 10000) PO = 100 or larger is a better choice. With millions of markers, as it is the case in GWAS, values as large as 10000 may be necessary.

Interpreting the results

After verifying that the chain has converged using the methods described in subsection "Evaluating Convergence" (see below), then we can interpret the results. For this we focus on the *_fst.txt file, which includes six columns.

The first column is the SNP index or ID as given in the input file. The next three (prob, log10(P0), and qval) are related to the test of local adaptation, and therefore the model including the locus specific effect. These include the posterior probability for the model including selection, the logarithm of the posterior odds for the model with selection, and the q-value for the model with selection. The fifth column gives the size of the locus-specific effect (alpha parameter). The last one provides the locus-specific F_{ST} averaged over all populations.

In order to decide if a locus is a good candidate for being under the influence of selection we use the q_value.

Running BayeScenv

The method has been implemented in C++ and there are both GUI and command-line versions for all operating systems (OS X, Lynux, and Windows) but here we will only use the command-line version, which as opposed to the GUI versions, is exactly the same for all operating systems.

Input Files:

Environmental data:
Text file consisting of a single line with the environmental value corresponding to each population. Populations should be ordered as in the genetic input file.
Environmental values should represent an "environmental distance" between the value observed in a local population and some reference value.
The reference value should represent some sort of "ancestral environmental state". For example, humans colonised high-altitude habitats from the lowlands so the obvious reference is 0.
Very frequently, the mean across all populations is a good reference value. For example if one thinks about temperature, each species has an optimal temperature range. Thus temperatures above or below the thermal tolerance range could lead to

a selective pressure for local adaptation.

Genetic data:
The best approach is to prepare files with FSTAT, popgen, or Arlequin format. Then use PGDSpider
http://www.cmpg.unibe.ch/software/PGDSpider/
to convert them into GESTE/BayScan format, which is the same format used by bayescenv.

Output files

There is a large number of files that can be obtained by running bayescenv. The most important ones, which are also the ones printed by default, are:

***_Verif.txt**: you should use this file in order to verify that bayescenv read tye input files properly.

***.sel**: provide the trace of some of the parameters estimated by the model. This include the loglikelihood and the local population Fst's. This file is used to verify convergence of the chain.

***_fst.txt**: this file is used to interpret the results. In other words, identify the outlier loci.

Command line:

If you type "./bayescenv", you get the list of options with a brief explanation

```
--------------------------
| BayeScEnv usage:     |
--------------------------
-help        Prints this help
--------------------------
| Input                |
--------------------------
alleles.txt  Name of the genotypes data input file
-d discarded Optional input file containing list of loci to discard
-snp         Use SNP genotypes matrix
-env env.txt Name of the environmental factors input file
--------------------------
| Output               |
--------------------------
-od          Output file directory, default is the same as program file
-o alleles   Output file prefix, default is input file without the
             extension
-fstat       Only estimate F-stats (no selection)
-all_trace   Write out MCMC trace also for alpha and g parameters (can be
             a very large file)
--------------------------
| Parameters of the chain |
--------------------------
-threads n   Number of threads used, default is number of cpu available
-n 5000      Number of outputted iterations, default is 5000
-thin 10     Thinning interval size, default is 10
-nbp 20      Number of pilot runs, default is 20
-pilot 5000  Length of pilot runs, default is 5000
-burn 50000  Burn-in length, default is 50000
--------------------------
```

```
| Parameters of the model |
---------------------------
-upper_g 10   Upper bound for the Uniform prior for parameter g, default
              is 10.
-mean_alpha -1    Mean of alpha prior, default is -1
-pr_jump 0.1      Prior probability for non neutral models, default is 0.1
-pr_pref 0.5 Prior preference for the locus-specific model, default is
              0.5.
-lb_fis 0     Lower bound for uniform prior on Fis (dominant data),
              default is 0
-hb_fis 1     Higher bound for uniform prior on Fis (dominant data),
              default is 1
-beta_fis     Optional beta prior for Fis (dominant data, m_fis and sd_fis
              need to be set)
-m_fis 0.05   Optional mean for beta prior on Fis (dominant data with -
              beta_fis)
-sd_fis 0.01  Optional std. deviation for beta prior on Fis (dominant data
              with -beta_fis)
-aflp_pc 0.1  Threshold for the recessive genotype as a fraction of
              maximum band intensity, default is 0.1
---------------------------
| Output files             |
---------------------------
-out_pilot    Optional output file for pilot runs
-out_freq     Optional output file for allele frequencies
```

The output files we will use were generated with the following command lines:

Long run that lead to results that are reliable because convergence of the MCMC was achieved. The prior probabilities used for the RJMCMC are conservative so the number of false positives is likely to be low while power may be slightly decreased.

```
./bayescenv allelefreqs.txt -env environment.txt -o exampleLongRun -
n 5000 -thin 10 -nbp 20 -pilot 5000 -burn 50000 -pr_jump 0.1 -
pr_pref 0.5
```

Short run that did not attain convergence and leads to unreliable results

```
./bayescenv allelefreqs.txt -env environment.txt -o exampleShortRun
-n 5000 -thin 10 -nbp 10 -pilot 1000 -burn 1000 -pr_jump 0.1 -
pr_pref 0.5
```

Long run using non-conservative prior probabilities for the RJMCMC. The results may be reliable in the sense that convergence has been reached but also contain many false positives.

```
./bayescenv allelefreqs.txt -env environment.txt -o exampleLongRun -
n 5000 -thin 10 -nbp 20 -pilot 5000 -burn 50000 -pr_jump 0.5 -
pr_pref 0.5
```

Interpreting the results

Once we are confident that the output of bayescenv represents a sample from the posterior distributions (i.e. convergence has been reached), then we can interpret the results. As it was the case for BayeScan, we focus on the *_fst.txt file, which in this case includes seven columns:

The first three (PEP_g, qval_g, g) are related to the test of local adaptation, and therefore the model including the locus specific effect due to the environmental

variable. The three following ones are related to the locus-specific effect (alpha parameter). The last one provides the locus-specific Fst averaged over all populations.

The effect size of locus specific effects (g and alpha) are given by the third and sixth columns respectively. In order to decide if a locus is a good candidate for being under the influence of selection we use the q_value for the model including the environmental factor (qval_g). However, if there is a need to be very conservative we could use the posterior error probability (PEP_g). The interpretation of this test statistics is as follows:

The PEP (Posterior Error Probability) is an estimate of the posterior probability that the model including a focal parameter (g or alpha) is incorrect. It can be thought of as a local and more conservative version of the FDR. The smaller it is, the more confident we can be that it represents a true positive result.

The *q*-value is a less conservative test statistic and is directly related to the FDR. It is obtained by averaging over the PEPs of all loci that have a lesser PEP than the focal locus. As a consequence, the *q*-values are generally lower (i.e. less conservative) than the PEP for all loci (except for the smallest one).

The choice of statistic to use depends on what you plan to do with the results. PEP and q-values are complementary, and useful in different situations. The q-value is more appropriate when carrying out a genome scan aimed at identifying potentially interesting genomic regions. On the other hand, the PEP is a better choice if what we want to do is to obtain support for the hypothesis that a particular gene is under selection.

Evaluating convergence

We will use CODA, a R package that implements several convergence tests.

Load the package

```
> library(coda)
```

Read the data. To avoid problems further down the line, remove the first column (step number) from the dataset

```
> chain<-read.table("HsIMM-Uds100-2Short.sel",header=TRUE)
> chain<-chain[-c(1)]
```

Create a MCMC object with the correct thinning interval (in the exercises below is 10)

```
>chain<-mcmc(chain,thin=10)
```

Always start with a plot. A rough way of verifying convergence is to plot the trace and the posterior distribution of some of the parameters

```
> plot(chain)
```

Obtaining summary statistics for the MCMC chain. We can obtain mean, standard deviation, naïve standard error of the mean (ignoring autocorrelation of the

chain) and time-series standard error (taking into account autocorrelation) as well as quantiles of the sample distribution. For this we use the command

```
> summary(chain)
```

Is the sample from the MCMC large enough? An important step is to verify that the sample size used to estimate the posteriors is large enough. The effective sample size on which the parameter estimates are based can be much smaller than the sample size used (in our example 5000) because the MCMC algorithm explore the parameter space by moving in small steps. Thus, two consecutive values will be strongly correlated, that's why we use a thinning interval (in our examples is 10 iterations). We can check the correlation between sampled parameter values that our thinned chain uses for estimation:

```
> autocorr.diag(chain)
```

If there is some correlation then, the effective sample size will be smaller than the sample size used as input. We can verify this with the command

```
> effectiveSize(chain)
```

As you will see when doing the exercises, the effective size of the likelihood sample is much smaller than the input value (5000) while those of the Fst parameters are less affected by the correlation. It is clear why this is the case: the correlation decreases much more rapidly for the Fst's than for the likelihood.

Has the chain converged? Now we can use some more formal approaches for the detection of non-convergence.
A very simple test is Geweke's convergence diagnostic based on the comparison of the means of the first and last parts of a Markov chain. The diagnostic reports the z-scores for each parameter. For example, with $\alpha = 0.05$, the critical values of z are -1.96 and +1.96. We reject H_0 (equality of means => convergence) if $z < -1.96$ or $z > +1.96$.

```
> geweke.diag(chain, frac1=0.1, frac2=0.5)
```

Another popular test is Heidelberg and Welch's convergence diagnostic, which is also based on a single chain:

```
> heidel.diag(chain, eps=0.1, pvalue=0.05)
```

It is generally better to run more than one chain and then use tests that compare them. One popular test is the Gelman and Rubin's diagnostic. To use it you need first to combine two chains into a single mcmc list:

```
> combined = mcmc.list(chain1,chain2)
> plot(combined)
> gelman.diag(combined)
> gelman.plot(combined,ask)
```

The `gelman.diag` is based on a comparison of between and within chain variances. If the chains converged, these two variances should be equal. The output

of gelman.diag are the scale reduction factors for each parameter. A factor of 1 means that between- and within-chain variance are the same; larger values mean that they are fairly different, indicating convergence problems. The rule of thumb is that values below 1.1 or so are OK but, being a rule of thumb, this is not an extremely rigorous criterion.

A very useful graphic representation of this convergence test is given by the command `gelman.plot`, which shows you the trace of the scale-reduction over time (chain steps). Thus, it is very useful to see whether a low chain reduction is also stable (you will see that sometimes the factors go down and then go up again). More importantly it allows us to find out how long the burn-in should be in order to eliminate any bias that arises from the starting point of the chain. The length of the burn-in depends on both the method and the dataset. Values between 1000-10000 steps are fairly common and lead to low computational overhead but there are many population genetics methods that can require a much longer burn-in (sometimes millions of steps).

Exercises

As it is typical of all MCMC methods, bayescenv is computationally intensive and a single run can take several hours unless you use a workstation or a cluster with many processors. Thus, we won't be able to generate the output here. Instead, I have produced several output files using the command lines provided in the tutorial. Nevertheless, you can go ahead and start a short run to see how the run progresses for half an hour or so. In the meantime we will work with the output files I produced.

There are output files (in separate folders) for a total of four runs:

- Long Run with π = 0.10 and p = 0.5 (longRun-pi01p05)
- Short run 1 with π = 0.10 and p = 0.5 (shortRun1)
- Short run 2 with π = 0.10 and p = 0.5 (shortRun2)
- Long run with π = 0.50 and p = 0.5 (longRun-pi05p05)

All runs were generated using the same input files (also provided -see folder "Data").

1) Follow the tutorial and verify convergence of Short run 1 and Short run 2. Do the same for Long run with π = 0.10 and p = 0.5. Describe the results.

2) Using the results of the two long runs, answer the following questions (hint: you have to sort the files using the third column "qval_g"):
   a. How many SNPs can be considered as outliers if the target FDR = 0.01?
   b. How many would there be if we aim for FDR = 0.05?
   c. In both cases (a & b) what is the expected number of false positives for the chosen threshold?

3) Compare the results obtained with the two long runs when we choose to allow for a FDR = 0.05 (i.e. the proportion of false positives among all positives is 5%). Why do they differ?

4) You will find two outputs of BayeScan in the folder "BayeScan". They correspond to runs with prior odds 10 and 100. How many outlier loci do you identify if the target FDR is (a) 0.01, (b) 0.001, and (c) 0.0001. Explain how results change depending on the prior odds.