Using fhetboot

Sarah P. Flanagan

2017-01-05

A common way to identify loci putatively under selection in population genomics studies is to identify loci that have high differentiation Fst relative to their expected heterozygosity Ht, as described in Beaumont & Nichols (1996). However, the Fst-Ht distribution changes shape based on the demographics of the population, and some distribution shapes are less conducive to identifying outliers than others. The problem of different distribution shapes is exacerbated by the current implementation of analyses which assume the same distribution for all demographic parameters. **fhetboot** bootstraps across the existing dataset to generate confidence intervals that approximate the actual Fst-Ht distribution.

This package performs several tasks.
- Parses genepop files into R.
- Calculates allele frequencies, Ht, and Fst (three commonly-used Fst calculations).
- Bootstraps across loci to generate confidence intervals.
- Generates customizable Fst-Ht plots with confidence intervals.
- Identifies loci lying outside of the confidence intervals.
- Calculates p-values for each locus based on the bootstrapped distribution

Getting Started

Read in your data

The first step is to organize your data in the genepop format. If you've been using LOSITAN, the format is identical. For details on the genepop format, refer to this website. **fhetboot** accepts both haploid and diploid genepop files with alleles coded using either the 2- or 3-digit format.

```r
library(fhetboot)
gfile<-system.file("extdata", "example.genepop.txt",package = 'fhetboot')
gpop<-my.read.genepop(gfile)
```

```
##
## Parsing Genepop file...
##
##
## File description:  Numerical Analysis with Nm=10, N=1000, 75 Demes, sampling 5 populations.
##
## ...done.
```

This function outputs any descriptors you've included in the header of your genepop file. This function was adapted from adegenet.

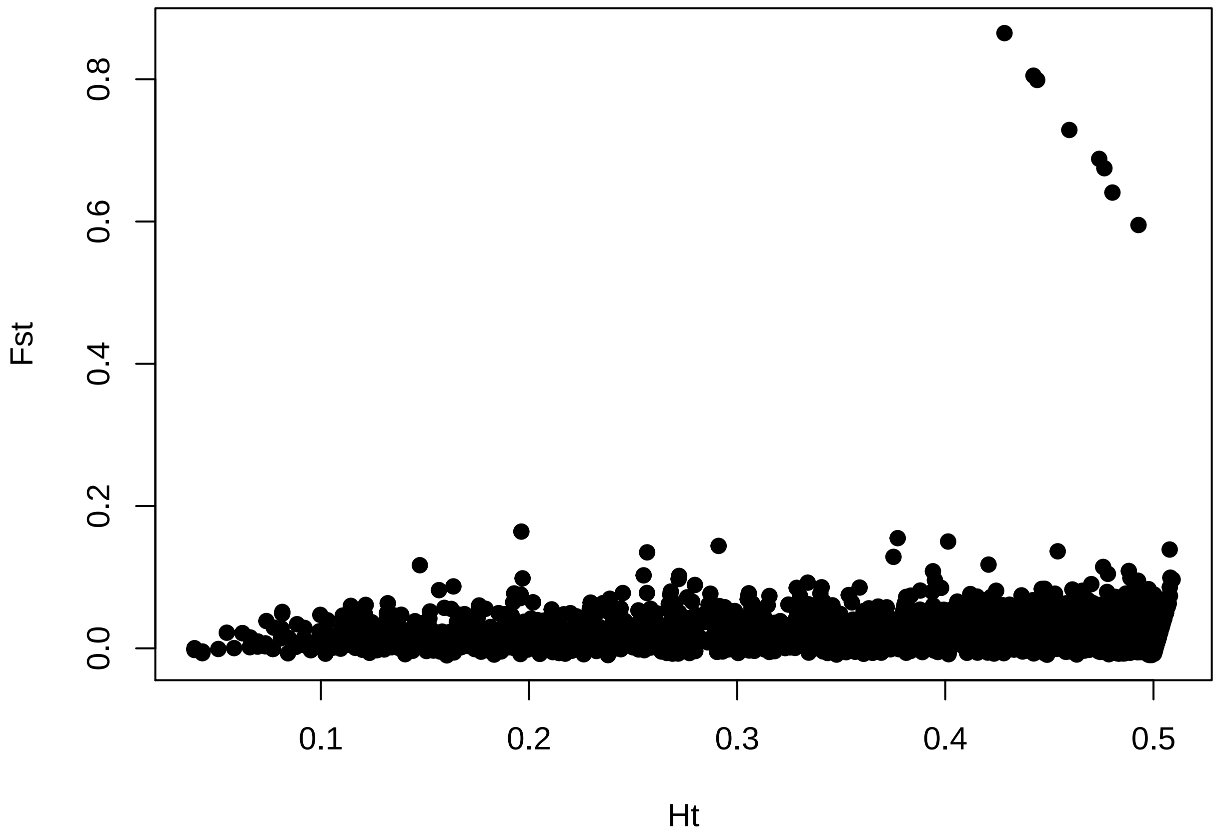
Calculate actual values

Before getting into any bootstrapping analyses, you must calculate the actual Fst and Ht values.

```r
fsts<-calc.actual.fst(gpop)
head(fsts)
```

```
##   Locus      Ht                  Fst
## 1  loc0 0.41154  0.00742370490147173
## 2  loc1 0.46314 -0.00866156054652011
## 3  loc2 0.49366   0.0432215850519948
## 4  loc3 0.49058   0.0118478601498701
## 5  loc4 0.42948   0.0276237272604974
## 6  loc5  0.2019   0.0646434628604019
```

```r
#Plot the actual values to see what your distribution looks like
par(mar=c(4,4,1,1))
plot(fsts$Ht, fsts$Fst,xlab="Ht",ylab="Fst",pch=19)
```



Since this distribution is not highly skewed, it should be fine for a bootstrap analysis. If you only have two demes (and your graph is skewed highly to the right), you might consider using alternative approaches to identifying outliers (e.g. BayeScan, BayEnv, PCAdapt).

Bootstrapping

The `boot.out` function conducts a single bootstrap run. We recommend running at least 10 bootstrap replicates. The bootstrap module is relatively slow, so if you want to verify that your data are correctly formatted and everything is going according to plan you can run it with a single replicate.

Testing it out

```r
boot.test<-fst.boot(gpop)
```

```
## [1] "Bootstrapping done. Now Calculating CIs"
```

```r
str(boot.test)
```

```
## List of 3
##  $ Fsts:'data.frame':    2000 obs. of  2 variables:
##   ..$ Ht : num [1:2000] 0.0393 0.0393 0.0507 0.0548 0.0548 ...
##   ..$ Fst: num [1:2000] -0.002382 0.000185 -0.000944 0.021983 0.021983 ...
##  $ Bins: num [1:19, 1:2] 0 0.075 0.1 0.125 0.15 0.175 0.2 0.225 0.25 0.275 ...
##   ..- attr(*, "dimnames")=List of 2
##   .. ..$ : chr [1:19] "4" "5" "6" "7" ...
##   .. ..$ : chr [1:2] "low.breaks" "upp.breaks"
##  $     :List of 1
##   ..$ CI0.95:'data.frame':   19 obs. of  4 variables:
##   .. ..$ Low   : num [1:19] -0.003 -0.008 -0.008 -0.008 -0.006 -0.005 -0.008 -0.007 -0.007 -0.004 ..
##   .. ..$ Upp   : num [1:19] 0.049 0.06 0.063 0.063 0.082 0.076 0.063 0.078 0.089 0.077 ...
##   .. ..$ LowHet: num [1:19] 0 0.075 0.1 0.125 0.15 0.175 0.2 0.225 0.25 0.275 ...
##   .. ..$ UppHet: num [1:19] 0.1 0.125 0.15 0.175 0.2 0.225 0.25 0.275 0.3 0.325 ...
```

```r
head(boot.test[[3]][[1]])
```

```
##      Low   Upp LowHet UppHet
## 1 -0.003 0.049  0.000  0.100
## 2 -0.008 0.060  0.075  0.125
## 3 -0.008 0.063  0.100  0.150
## 4 -0.008 0.063  0.125  0.175
## 5 -0.006 0.082  0.150  0.200
## 6 -0.005 0.076  0.175  0.225
```

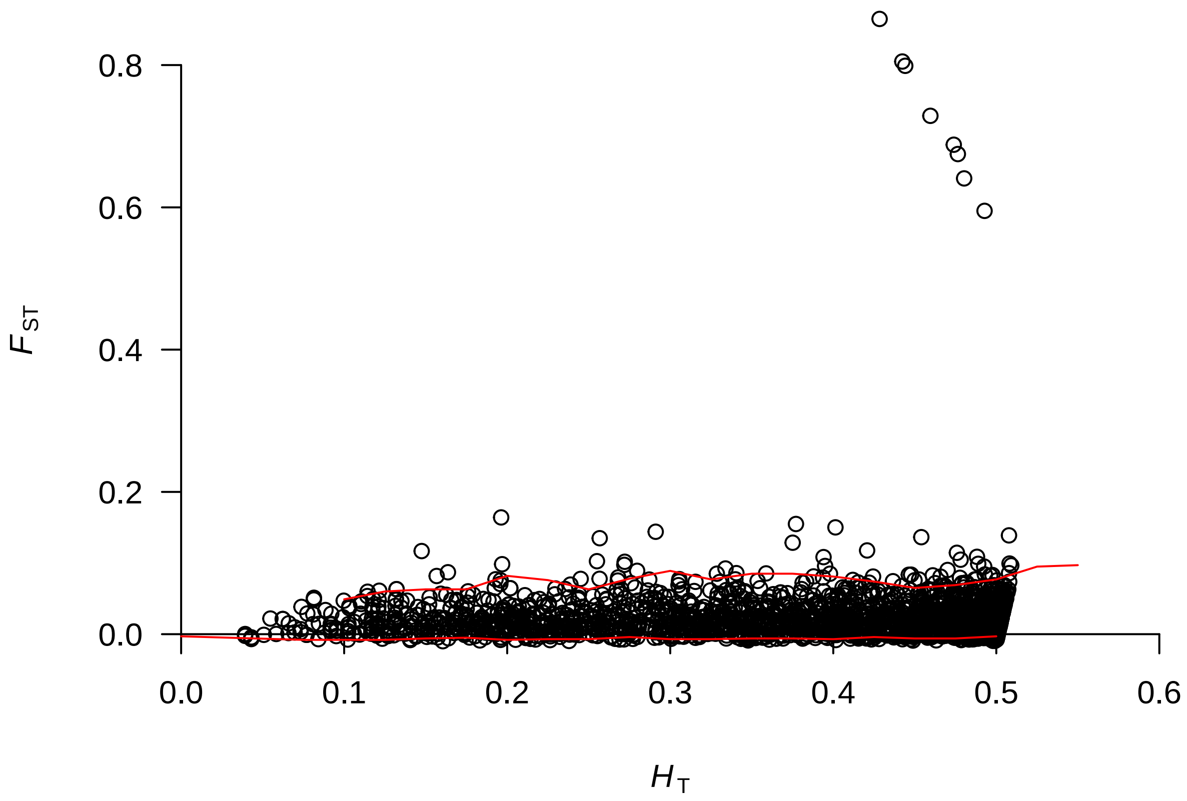
From the results of `str(boot.test)`, you can see that `fst.boot()` returns a list of three data frames: one with the bootstrapped Ht and Fst values, one with the bins from bootstrapping, and one with the Ht and Fst values for the upper and lower confidence intervals.

If you want to visualize these results, you can use `plotting.cis`. Plotting.cis requires the raw datapoints (`fsts`) and either a list with the confidence intervals or a list of multiple bootstrap replicates. Since we only ran a single bootstrap replicate in this case, we'll give `plotting.cis` a list.

```r
#plot the results
ci.list<-ci.means(boot.test[[3]])
head(ci.list)
```

```
##          low   upp LowHet UppHet
## 0.1   -0.003 0.049  0.000  0.100
## 0.125 -0.008 0.060  0.075  0.125
## 0.15  -0.008 0.063  0.100  0.150
## 0.175 -0.008 0.063  0.125  0.175
## 0.2   -0.006 0.082  0.150  0.200
## 0.225 -0.005 0.076  0.175  0.225
```

```r
par(mar=c(4,4,1,1))
plotting.cis(df=fsts,ci.df=ci.list,make.file=F)
```



Clearly, these confidence intervals are not ideally smoothed (since they only represent a single replicate), but you can see that they generally follow the shape of the distribution.

The function `smooth.cis` can also be used to smooth the confidence intervals. All this does is remove some of the intermediate points from the confidence intervals.

fhetboot also contains the option for calculating p-values. To calculate p-values, run `p.boot` on the bootstrapping output. For this function to work, you must have run `fst.boot` more than once. It takes the data.frame of actual Fst values and either the bootstrapping output or the output from running `fst.boot.means`. You can then use a false discovery rate or p-value correction to coorect for multiple testing.

```r
#run bootstrapping 10 times
boot.out<-as.data.frame(t(replicate(10, fst.boot(gpop))))
```

```
## [1] "Bootstrapping done. Now Calculating CIs"
## [1] "Bootstrapping done. Now Calculating CIs"
## [1] "Bootstrapping done. Now Calculating CIs"
## [1] "Bootstrapping done. Now Calculating CIs"
## [1] "Bootstrapping done. Now Calculating CIs"
## [1] "Bootstrapping done. Now Calculating CIs"
## [1] "Bootstrapping done. Now Calculating CIs"
## [1] "Bootstrapping done. Now Calculating CIs"
## [1] "Bootstrapping done. Now Calculating CIs"
## [1] "Bootstrapping done. Now Calculating CIs"
```

```r
boot.pvals<-p.boot(fsts,boot.out=boot.out)
head(boot.pvals)
```

```
##       loc0       loc1       loc2       loc3       loc4       loc5
## 0.47560976 0.12089810 0.53886010 0.58376511 0.90073529 0.04255319
```

```r
boot.cor.pvals<-p.adjust(boot.pvals,method="BH")
boot.sig<-boot.cor.pvals[boot.cor.pvals <= 0.05]
```

Running the actual analysis

The simplest way to run the analysis is to use the `replicate` function. I recommend running about 10 bootstrap replicates. I tested 10, 100, and 1000 replicates and they all yielded qualitatively similar results, so since the `fst.boot` function is rather slow you can get away with using a minimum of 10 replicates. You might want to test out different numbers of replicates for yourself, if you want to make sure you're accurately representing your data.

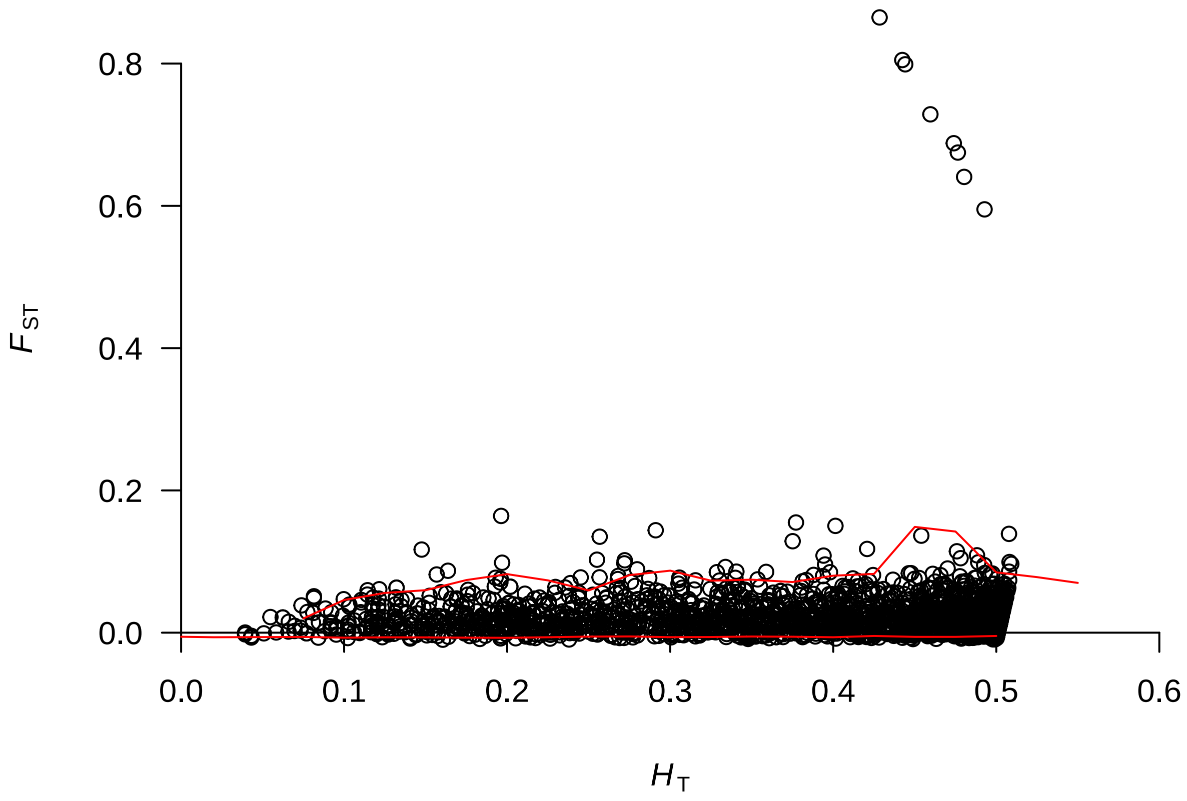
```r
boot.out<-as.data.frame(t(replicate(10, fst.boot(gpop))))
```

```
## [1] "Bootstrapping done. Now Calculating CIs"
## [1] "Bootstrapping done. Now Calculating CIs"
## [1] "Bootstrapping done. Now Calculating CIs"
## [1] "Bootstrapping done. Now Calculating CIs"
## [1] "Bootstrapping done. Now Calculating CIs"
## [1] "Bootstrapping done. Now Calculating CIs"
## [1] "Bootstrapping done. Now Calculating CIs"
## [1] "Bootstrapping done. Now Calculating CIs"
## [1] "Bootstrapping done. Now Calculating CIs"
## [1] "Bootstrapping done. Now Calculating CIs"
```

`boot.out` is a data.frame with three elements: the bootstrapped values (Fsts), the bins used in the bootstrapping (Bins), and a list of the upper and lower confidence intervals (V3). Each of these elements is a list of data.frames, with one data.frame per bootstrap replicate. Use `str` to visualize the output if you're confused.

Now we want to generate a plot with the confidence intervals. Again, `plotting.cis` will do the trick. We can also use the `find.outliers` function to pull out a data.frame containing the loci that lie outside of the confidence intervals.

```r
par(mar=c(4,4,1,1))
plotting.cis(df=fsts,boot.out=boot.out,make.file=F)
```



```r
outliers<-find.outliers(fsts,boot.out=boot.out)
head(outliers)#the CI outliers
```

```
##          Locus                  Ht                  Fst
## 38       loc37 0.0623400000000001   0.0213329963089365
## 863     loc862 0.0738200000000001   0.0383378625047556
## 1308   loc1307 0.0547800000000001   0.0219832498036245
## 1675   loc1674 0.0623399999999998    0.021332996308933
## 1706   loc1705             0.04306 -0.00681689162878519
## 228     loc227 0.0814200000000002   0.0487036912976492
```

Customizing the Figures

The default `plotting.cis()` output may not be ideal for publication. Luckily, the function `plotting.cis()` has several built-in options for customizing the plot.

The data you use

As demonstrated in the two cases above, `plotting.cis()` requires the original data and one of two things with the confidence intervals.
- The results from multiple bootstrap replicates (as generated by the above example), and it then calculates the mean confidence intervals from those. This is specified by `plotting.cis(boot.out=<name>)`.
- A `ci.list`, which is actually a data.frame with Ht values as row names and two columns: low, and upp. These header names are requried for it to work.

One of these two things is required or `plotting.cis()` will fail.

If your actual data (`df=<name>`, or `fsts` in the above examples) have different column names, you can specify those using `plotting.cis(Ht.name=<name>)` and `plotting.cis(Fst.name=<name>)`. Otherwise, the defaults are `plotting.cis(Ht.name="Ht",Fst.name="Fst")`.

The look of the graph

Several aspects of the graph can be controlled through `plotting.cis()`: the color of the confidence interval lines and the shape of the points. These are controlled by `ci.col` and `pt.pch`.
The defaults are: `plotting.cis(ci.col="red", pt.pch=1)`.
The default CI color is red and the defualt point shape is open circles (`pt.pch=1`). You can also color-code some loci (for instance ones with significant p-values) using `sig.col`. The default setting for `sig.col` is to be identical to `ci.col`. The default setting is to smooth the confidence intervals with the settings `smooth.ci=TRUE, smoothing.rate=0.025`.

Saving the graph to a file

In the above examples, you may have noticed that `plotting.cis()` always contained the command `make.file=F`. This command allows you to automatically save the graph to a file or to print it to the default device in R. If `make.file=TRUE`, then the function generates a *.png file. The default file name is "OutlierLoci.png", but this can be changed using `file.name`. If you choose to designate a file.name, it must contain the ".png" extension. For example:

```
plotting.cis(df=fsts,boot.out=boot.out,make.file=T,file.name="ExampleOutliers.png")
```

Other Functions

I just want to take a moment to discuss what the other functions in **fhetboot** do and some other ways to use the proram.

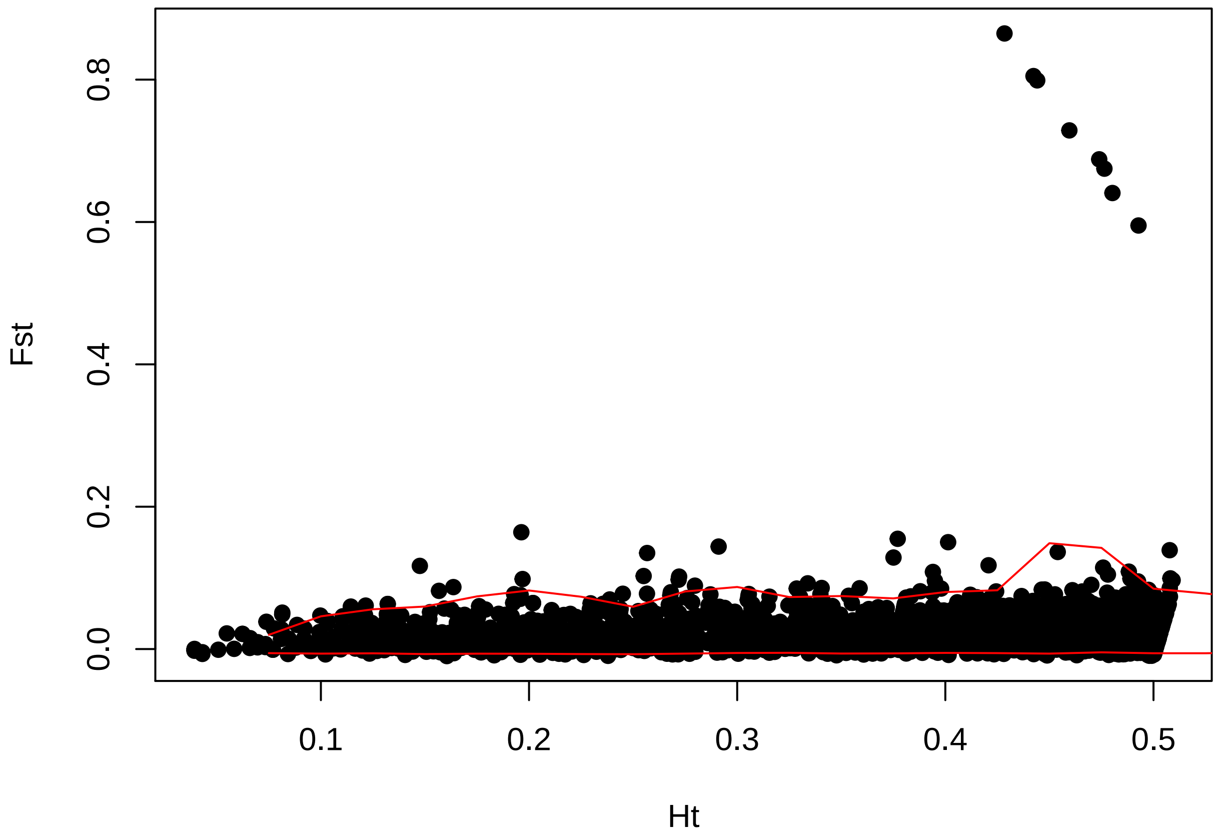
Saving the data and plotting it yourself.

Although `plotting.cis` can be a useful tool, it is possible to save your confidence intervals and generate your own plots. First, you use the function `ci.means()` to calculate the mean confidence intervals across all of the bootstrap replicates, and then you can generate a plot and add the confidence intervals using `points()`.

```
#calculate means
boot.out.ci<-ci.means(boot.out[[3]])

#create a data.frame of confidence intervals
cis<-as.data.frame(do.call(cbind,boot.out.ci))
colnames(cis)<-c("low","upp")
cis$Ht<-as.numeric(rownames(cis))

#plot
par(mar=c(4,4,1,1))
plot(fsts$Ht, fsts$Fst,pch=19,xlab="Ht",ylab="Fst")
points(cis$Ht,cis$low,type="l",col="red")
points(cis$Ht,cis$upp,type="l",col="red")
```

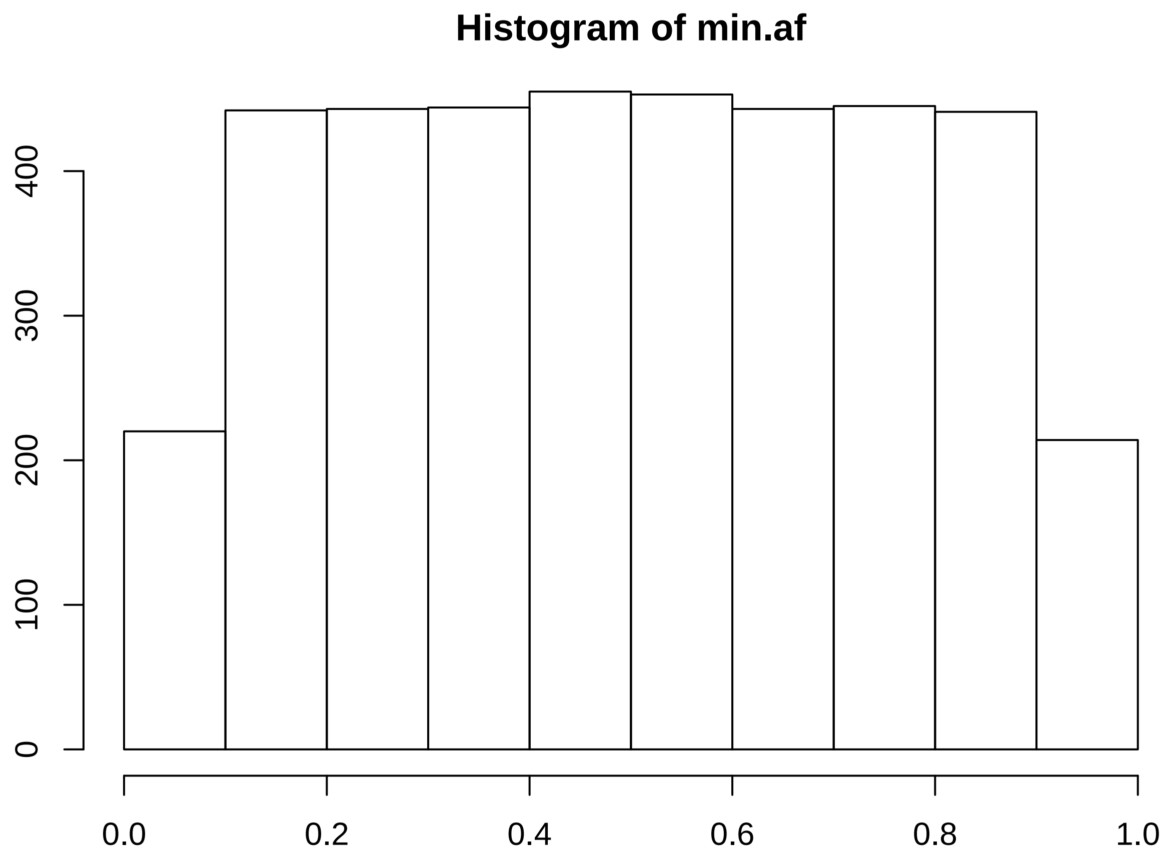


Look at the distribution of allele frequencies

The analyses in **fhetboot** use the function `calc.allele.freq` to calculate allele frequencies. If you're interested in examining the allele frequency distribution in your dataset, you can use this function on your actual data.

```
af.actual<-apply(gpop[,3:ncol(gpop)],2,calc.allele.freq)

#extract the minimum allele frequency for each locus
min.af<-unlist(lapply(af.actual,min))
par(mar=c(2,2,2,2))
hist(min.af)
```



Conclusion

Hopefully this package will be a useful tool for population geneticists and molecular ecologists. It's important to consider the assumptions of the tests you use as well as remembering that statistics should be used to describe your dataset. Use your common sense about when to use different methods and how to implement them. Good luck!

If you run into any problems, find any bugs, or have other comments on fhetboot please contact me: spflanagan.phd@gmail.com.
