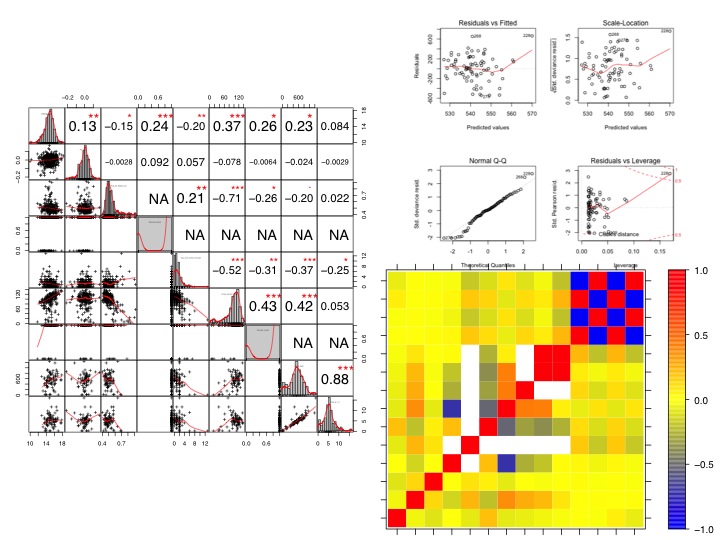
***HeFPipe*: An analytical Pipeline for Heterozygosity-Fitness Correlations**

Version 1.1

****

**Author:**

Mark A. Fisher

Department of Genetics

University of Georgia

Email: [mark.aaron.fisher@gmail.com](mailto:mark.aaron.fisher@gmail.com)

Software URL: [https://github.com/Atticus29/*HeFPipe*\_repos](https://github.com/Atticus29/HefPipe_repos)

Video Tutorial URL: <http://www.youtube.com/playlist?list=PLv-e9CNPZr-o34dIwUKi-Eew-t6A643tX>

Table of Contents

About *HeFPipe* 3

Citing *HeFPipe* 3

Installation 4

System Requirements 4

Dependencies 4

Installing *HeFPipe* 4

The Input Files 4

Addresses File 5

Allele Reports Directory 5

Loci to Include File 5

Loci to Exclude File 5

Pipeline Directory 6

Samples to Exclude File 6

Trait File 6

Some Important Notes About the Input 7

Allele Reports 7

Sample, Locus, and Trait Names 7

General Considerations 8

Using *HeFPipe* 8

Running the Example Data 8

Preparing User-Provided Data 9

Processing the Allele Reports for *GenePop* 9

Running *GenePop* on the Web and Feeding the Output Back into *HeFPipe* 10

Generation of *RMES*-, *GEPHAST*-, and *Rhh*-Related Files 11

RMES **Error! Bookmark not defined.**

GEPHAST **Error! Bookmark not defined.**

Rhh **Error! Bookmark not defined.**

Correlations 13

Testing for Single-Locus Associations (SLAs) with an F-ratio Test 15

Running Regression Analyses 17

Full Model Regressions 17

Data Subset Regressions 18

User-Generated Data Subset Regressions 19

Converting Regression Results into Spreadsheets 20

References 21

**Please read this manual before beginning analysis.**

**Disclaimer**

Although *HeFPipe* has been tested and, to the best of our knowledge, works without problems, the author take no responsibility for any bugs, errors, or any harm or damage caused by the software or the documentation. The software and the documentation come without any warranty; without even the implied warranty of merchantability or fitness for a particular purpose. Use the software and the documentation at your own risk.

# About *HeFPipe*

*HeFPipe* is a script written in *Python* and *PypeR* that conducts analyses typically performed in heterozygosity-fitness correlation (HFC) studies. It also tests for evidence of single-locus effects on a phenotype(s). More specifically, *HeFPipe* takes input in the form of allele reports from the microsatellite genotype-calling software, *GeneMarker*, and reconfigures the data into *GenePop*(Raymond & Rousset, 1995), *Rhh* (Alho, Välimäki, Merilä, Valimaki, & Merila, 2010), *RMES* (David, Pujol, Viard, Castella, & Goudet, 2007), and *GEPHAST* (Amos & Acevedo-Whitehouse, 2009) formats. The script is also equipped to re-format the *output* from *GenePop* on the Web (option 5) and *Rhh* into csv spreadsheets, where the data are reported alongside other statistical characteristics of the data set. The *HeFPipe* script accommodates user-provided lists of markers to include and exclude from analyses, a list of samples to exclude from analyses, and a spreadsheet containing trait/phenotype values on which to perform the HFCs and to look for single-locus effects. With regards to these last two analyses, *HeFPipe* is equipped to run generalized linear models (GLMs), and the user is able to designate the error distribution and link functions relevant to the trait(s). Single-locus associations (SLAs) are explored using an F-ratio test (methodologies described in (Szulkin, Bierne, & David, 2010)) and using *GEPHAST* (Amos & Acevedo-Whitehouse, 2009). Correlations (both Pearson and Spearman) among the traits provided are reported in several different formats (as text, spreadsheets, and images); significance tests are conducted on these correlations, and the *P*-values (both adjusted and unadjusted for multiple comparisons) are also reported in the various formats.

# Citing *HeFPipe*

A computer note about *HeFPipe* has been published in the journal, Molecular Ecology Resources. If you use *HeFPipe*, the package should be cited as:

Citation here

# Installation

## System Requirements

1. *HeFPipe* was designed to work on Mac OS X operating systems.
2. 32.1 MB free hard disk space.

## Dependencies

1. *HeFPipe* is run through a command-line terminal and requires basic knowledge of command-line syntax. There are many basic tutorials available online, such as this one: <http://cs.colby.edu/maxwell/courses/tutorials/terminal/>.
2. Download & install R v. 2.15.1. (<http://www.r-project.org/>)
3. Install R packages ‘reshape’, ‘*Rhh*’, ‘ggplot2’, ‘pgirmess’, ‘lattice’, ‘MASS’, and ‘PerformanceAnalytics’. Once R is installed, these can be installed from the Package Installer option from the Packages & Data menu (make sure to click the “install dependencies” box).
4. Download and install *Python* v 2.7.3—not v 3.2.3!— ([http://www.*Python*.org/getit/](http://www.python.org/getit/)). A version of pyton 2.x usually comes pre-installed on Mac OS X operating systems.
5. *Python* packages os, re, csv, pickle, copy, math, collections, and itertools should already be pre-installed in *Python*. *PypeR* v. 1.1.1 will install automatically when *HeFPipe* is installed below.
6. Download the *GEPHAST* macro (Amos & Acevedo-Whitehouse, 2009) for Microsoft Excel (<http://www.zoo.cam.ac.uk/zoostaff/meg/amos.htm>).
7. Microsoft Excel must also be installed for the *GEPHAST* component.

## Installing *HeFPipe*

1. Download the HeFPipe package from GitHub ([https://github.com/Atticus29/*HeFPipe*\_repos](https://github.com/Atticus29/HefPipe_repos)).
2. Navigate (using ‘cd’ in command-line) to the directory containing the package you just downloaded. You should see a file called, ‘setup.py’. Type, ‘python setup.py install’ in command-line. This should install the pyper.py, HefPipe\_modules.py, and HefPipe.py scripts onto your machine.

# The Input Files

In order to run *HeFPipe*, a few input files are required; a template of each of these files comes with the example data. These files are:

## Addresses File

A csv (comma-separated values) file (called, ‘addresses.csv’ in the example data set) containing the paths for all of the input files (see Running the Example Data). Note that you should not include quotation marks for the paths designated in this file.

## Allele Reports Directory

A directory (called ‘allele\_reports’ in the example data set) containing all of the allele reports from *GeneMarker* in csv format. Note: The allele reports need to be in the “Marker Table” style (as opposed to “Peak Table” style). As discussed below, the first file in this directory must contain *all* samples listed in the other files. In other words, the samples in all other allele reports are permitted to be subsets of the samples included in this first allele report. If this is not possible, see the “Some Important Notes About the Input” section below for a solution.

## Loci to Include File

A csv file (called ‘keeplist.csv’ in the example data set) containing a list of every microsatellite locus that will be used in the current run of the pipeline. Loci are listed in the first column of the spreadsheet, one locus to each row. Note: *HeFPipe* will throw errors and arrest if a locus with no polymorphism is included in keeplist.csv.

## Loci to Exclude File

A csv file (called ‘monolist.csv’ in the example data set) containing a list of every microsatellite locus that is contained in the allele reports that will *not* be used in the current run of the pipeline. This may be because the loci are monomorphic or difficult to score. This is intended to save the user time by not having to omit these data manually from all of his/her allele reports either during or after scoring.

## Pipeline Directory

The output files of the pipeline will be saved in this directory (called ‘pipeline’ in the example data set) or in subdirectories within this directory. The user creates this directory manually and provides the path in addresses.csv.

## Samples to Exclude File

A csv file (called ‘rejected\_samples.csv’ in the example data set) containing a list of every sample that will be excluded from the current run of the pipeline, one sample per row in the first column (no header). In our experience, these excluded samples have included samples that ended up being triploid based on their microsatellite genotypes or for which the DNA extraction failed. We can imagine any number of reasons to exclude certain samples for various runs of the pipeline, and the samples to exclude file allows for this exclusion to be accomplished relatively easily. Note: triploid scores for loci in keeplist will be flagged as problematic, and users will be asked to delete the extra column in the relevant allele report before moving on. The user may choose either to remove only the third column or to remove the third column and delete the genotype of the offending individual at the entire locus. As long as that individual/sample is listed in the samples to exclude file, these actions will have the same outcome. It is also important to note that samples can be listed in the rejected samples list for reasons pertaining to either its genotype(s) or trait(s); the script will make sure that the rejected samples will be excluded from relevant analyses either way. This allows users to avoid the tedium of making sure that samples are disabled from every single allele report. *It may require some thought about which samples should be rejected outright, and which should simply contain missing fitness and/or genotype data.* Finally, the user should bear in mind that there is a portion of the pipeline that allows you to generate and run regression models on subsets of the data (e.g., individuals stressed by viral infection).

## Trait File

In order to conduct a heterozygosity-fitness correlation, at least one trait must be provided, although the pipeline accommodates any number of traits. This csv file (called ‘acceptor.csv’ in the example data set) contains these fitness data (with sample IDs—as numbers—as the first column), one trait per column with headers. Notes: 1) The trait data is permitted to contain samples that are not present in the genotype data and vice versa. Missing data or even entire missing samples (i.e., a row of data with only the sample ID) are also permitted and should be specified as “NA”. 2) For optimal results when the correlation heat map and correlation charts are generated, the column headers (i.e. trait names) in this spreadsheet should be as short and informative as possible.

## Some Important Notes About the Input

### Allele Reports

1. The first allele report in the allele report directory (when sorted alphabetically) must contain every sample used in all of the other allele report files. If this cannot be accomplished by renaming files, a dummy allele report can be made using one of the other allele reports as a template. This dummy allele report would contain every sample and each sample’s corresponding genotype at a dummy locus (these genotypes would be fake), and that dummy locus would then be added to the loci to exclude file. Finally, give the dummy allele reports a file name that would ensure that it is the first file (in alphabetical order) in the directory (see example data set 3\_With\_dummy\_allele\_report for an example).
2. All other allele report files must contain at least a subset of the samples that will eventually be included in the analyses.

### Sample, Locus, and Trait Names

1. Don't name the same samples differently across different allele report files.
2. Blanks that you don't want to include and are not in the rejected sample list have to contain the string "Blank" or "blank" in them. Similarly, samples that you want included in the analysis cannot contain the string “Blank” or “blank” in them.
3. Samples should be named according to the following convention: #+OtherInformation. In other words, designate each sample with a number. If the sample names must contain other information for some other purpose in your analysis, this is fine, but the pipeline will ignore all symbols including and beyond the first non-digit character. While the sample numbers must be consistent across allele reports (i.e. correspond to the same sample), the other information is allowed to be different.
4. Don't name any of your loci 'Pop', 'pop', or 'POP'. This will interfere with *GenePop* formatting.
5. Locus names cannot contain “-“ because this symbol will be used as a subtraction operator when the analyses are passed to R.
6. Locus names cannot contain spaces.
7. Similarly, don’t name any of your traits or loci ‘done’ or ‘Done’. This will interfere with the regression and single-locus association analyses of the pipeline.
8. Trait names cannot contain spaces.
9. Your population cannot have the same name as any of your loci.

### General Considerations

1. Heterozygosity-fitness associations are generally not appropriate to conduct on multiple populations (Slate & Pemberton, 2006). *HeFPipe* is accordingly not equipped to accommodate multi-population data.
2. The current version of *HeFPipe* is only configured to analyze diploid data. Make sure that you have no loci in any of your allele reports that contain only one column of scores.

# Using *HeFPipe*

## Running the Example Data

1. A directory called, “Example\_data” is included with *HeFPipe*. In it, you will find three subdirectories (‘1\_Before\_the\_pipeline\_is\_run’, ‘2\_After\_the\_pipeline\_is\_run’, and ‘3\_With\_dummy\_allele\_report’). Each of these subdirectories contains five csv files (‘acceptor.csv’, ‘addresses.csv’, ‘keeplist.csv’, ‘monolist.csv’, and ‘rejected\_samples.csv’) and two directories (‘allele\_reports’ and ‘pipeline’). The ‘allele\_reports’ directory will contain several files that are output from *GeneMarker* (converted to csv format), and ‘pipeline’ will also contain a few example files for how downstream output should be converted into input for other parts of the pipeline.
2. Begin with the ‘1\_Before\_the\_pipeline\_is\_run’ directory. Open the ‘addresses.csv’ file. Place the example files referenced in the rows of this csv file in the locations you desire on your computer, and update the paths in column B of ‘addresses.csv’ accordingly.
3. In terminal, type, ‘python HeFPipe.py’, and follow the prompts of the program (see Processing the Allele Reports for *GenePop* and subsequent sections for specific details). When you have completed the pipeline, the results should be the same as the results found in the ‘2\_After\_the\_pipeline\_is\_run’ subdirectory.

## Preparing User-Provided Data

1. Once you have run the pipeline using the example dataset, simply provide and/or edit the five csv files described in ‘Running the Example Data’ using your own data, replace the allele reports (converted to csv format) in the Allele Reports Directory with your own. Make sure that these are in csv format. One way to batch convert excel files into csv files easily on Mac OS X is with the software, Excel to CSV Converter: (<http://www.macupdate.com/app/mac/36172/excel-to-csv-converter>).
2. In terminal, navigate to the directory containing ‘HeFPipe.py’ and ‘HeFPipe\_modules.py’. Type, “python HeFPipe.py”, and follow the prompts of the program (see Processing the Allele Reports for *GenePop* and subsequent sections for specific details)

## Processing the Allele Reports for *GenePop*

1. Upon execution *HeFPipe*.py, you are prompted to type the path of the addresses.csv file. Type the path and press enter (or simply drag the file into the terminal window, being mindful of spaces that can get added when doing it this way).
2. The pipeline will then output the name of the first allele report being processed. If that allele report contains missing data that are not designated as, “\*\*”, it will inform the user which sample(s) bears the missing data.
3. Step 2 will repeat for each allele report in the allele reports directory. You will notice that as each allele report is processed, corresponding edited allele report files are generated and stored in the pipeline directory.
4. Once all of the edited allele reports have been made, *HeFPipe* will report sample IDs that are missing from each report. The user may expect some of these to be missing from certain reports or not (for instance, this is expected in the example data set provided). The script waits for the user to indicate that these missing samples are acceptable or to abort the pipeline (cmd+C in Terminal) and adjust the allele reports before re-running the pipeline. When ready, type “y” and press enter.
5. The pipeline then generates a file called, “missing.csv”, which tells users whether any of the loci expected to end up in the final, processed dataset are not ending up there (either because they are absent from the allele reports or are otherwise causing errors). If any loci appear in “missing.csv”, inspect the relevant allele report(s) for mistakes, correct them, and re-run the pipeline.
6. At this point, the pipeline also generates a file called, “final\_output.csv” that contains the genotypes of all of the relevant samples at all of the relevant loci. Samples on the rejected samples list will be missing from this dataset. Missing genotypes will be denoted with “\*\*”. Note that this file is *not* sorted by ID.
7. You are then prompted for the name of the population. Enter the name (e.g., “Oglethorpe” for the example data set) and press enter.
8. The pipeline then generates a “final\_output\_*GenePop*.txt” file, which is ready for analysis in the program *GenePop*.

## Running *GenePop* on the Web and Feeding the Output Back into *HeFPipe*

1. At this point the pipeline asks you whether you want to calculate the effective number of alleles (with the help of *GenePop*). If you do, type “y” or “yes”. You will then have to submit the data from the final\_output\_*GenePop*.txt file to *GenePop* on the Web, option 5, suboption 1 (“Basic Information, FIS, and gene diversities”: [http://*GenePop*.curtin.edu.au/*GenePop*\_op5.html](http://genepop.curtin.edu.au/genepop_op5.html)), opting for the “HTML-Plain Text” format of output from *GenePop*. When that output is generated, copy it from the browser into a plain text file and save the text file (“opt5.txt” in the pipeline directory of the example data set). Note: make sure the *entire* HTML output gets pasted into the text file.
2. The pipeline then prompts the user to type the path of this file, which you can type manually or paste into the terminal by dragging the file into the terminal window as described earlier (again, be mindful of spaces at the end of the path name created by this second option).
3. Once the path has been entered, the pipeline generates a csv file called, “allele\_freqs.csv” that contains allele frequencies and FIS indeces for each allele at each locus in the data set. It then generates a separate csv file called “Effective\_alleles\_per\_locus.csv”, which contains the actual number of alleles and the effective number of alleles (Frankham, Ballou, & Briscoe, 2010). Finally, the pipeline generates a csv file called, “H\_obs\_and\_H\_exp.csv”, which contains the observed and expected heterozygosities of each locus in the data set (calculated by *GenePop*).

## Generation of *RMES*-, *GEPHAST*-, and *Rhh*-Related Files

### *RMES*

1. The pipeline then uses the ‘final\_output\_*GenePop*.txt’ file it generated earlier to create another text file formatted for Patrice et al.’s *RMES* (robust multilocus estimate of selfing) software (David et al., 2007). Note that *RMES* only operates on Windows platforms, so the *RMES*-ready file may need to be opened in Microsoft Word and transferred between operating systems in that format to preserve format integrity.

### *GEPHAST*

1. The pipeline also generates a directory called, “*GEPHAST*\_ready” that contains csv-formatted files for each trait column in the trait file, formatted in a way that the *GEPHAST* macro(Amos & Acevedo-Whitehouse, 2009) in Microsoft Excel can be run on it. Run *GEPHAST* on as many or as few of these csv files as desired. After each test has been completed, save the csv files containing the *GEPHAST* results into a separate directory that you’ve generated (this directory is called “*GEPHAST*\_tested” in the example data, and it contains example *GEPHAST* results).
2. You are then asked whether you want to generate a list of *GEPHAST* *P*-values for the traits upon which *GEPHAST* tests have been run. Type, “y” or “yes”, and you are prompted once more to ensure that *GEPHAST* tests have been conducted and the results have been placed in a designated directory. Type any key to proceed.
3. At this point, the pipeline asks you to supply the path of the directory containing the *GEPHAST* results. You can either type these manually or drag the directory to the terminal window. Either way, *make sure that the path ends with a backslash* (for example, “/Users/mf/Desktop/pipeline/*GEPHAST*\_tested/”). Press enter.
4. The pipeline next generates two files: ‘*GEPHAST*\_pvals.csv’ and ‘*GEPHAST*\_p\_adjusted.csv’. These contain the *P*-values or adjusted *P*-values, respectively, for tests for significant single-locus associations for each locus at each trait for which there was a result file in the designated *GEPHAST* result directory. *P*-values are listed in every even column, while the loci are listed in every odd column. Trait names serve as headers for the odd columns. *P*-values were adjusted to account for multiple comparisons with the Benjamini-Hochberg correction assuming a 5% false discovery rate using the p.adjust function in R. Note that the multiple comparisons in the pipeline were applied at the trait level, not at the global level. Note also that these files are not sorted by *P*-value.

### *Rhh*

1. The pipeline then automatically converts the data into a format that it can then pass to the R package *Rhh* (Alho et al., 2010); this file is called, “*Rhh*\_ready.txt”.
2. The pipeline also generates three other files at this point: 1) a file called, ‘heterozygotes\_and\_homozygotes .csv’ in which all of the diploid genotypes from ‘final\_output.csv’ have been converted into 1 if they are heterozygous, 0 if they are homozygous, and ‘NA’ if the data are missing (note that information for each locus has been compressed into one column each in this file), 2) a file called, ‘MLH\_calculated.csv’ that contains the MLH values for each sample in ‘final\_output.csv’, and 3) a file called ‘Loci\_typed\_per\_individual.csv’ that contains the number of loci at which each individual was successfully genotyped.
3. You are then prompted to run the *Rhh* script, which conducts the heterozygosity-heterozygosity correlation (HHC) test for identity disequilibrium/inbreeding (Balloux, Amos, & Coulson, 2004). As it is currently configured, the script runs through 250 iterations (see *Rhh* documentation for more information), but it is possible to change this value in ‘*HeFPipe*\_modules.py’. Type “y” or “yes” and press enter.
4. The *Rhh* script then generates several documents: 1) ‘number\_of\_alleles.txt’, which displays the number of alleles for each locus, 2) ‘*Rhh*\_test\_output .txt’ and 3) ‘*Rhh*\_test\_output.txt\_csv\_converted.csv’, which are a text file or a csv file, respectively, of the IR, SH, and HL values of each sample, 4) ‘hhc\_plot.pdf’, which is a plot of the distribution of correlation coefficients generated by the HHC test, and 5) ‘mean\_and\_corr\_probs.txt’, which displays the mean correlation coefficient and 95% confidence intervals for the HHC test.
5. The pipeline also generates a file called, ‘MLH\_output.csv’, which combines MLH, IR, SH, and HL values (calculated previously in steps 2 and 4) with the user-provided trait file. This ‘MLH\_output.csv’ file is then used downstream by the pipeline for regression analyses.
6. After exiting the *Rhh* module, the pipeline creates a file called, ‘number\_of\_samples\_per\_locus.csv’ in the pipeline directory. This file is spreadsheet that contains two columns: the first column lists all of the loci in the loci to include file, and the second column lists the number of samples at which each of those loci were successfully scored.

## Correlations

1. The pipeline also generates a correlation matrix heat map of all of the traits (and MLH, IR, SH, and HL) called, ‘Heat\_map.pdf’, which it stores in a new subdirectory of the pipeline directory called, ‘Correlations’. It also generates a file called, ‘all\_correlations.csv’, which contains Pearson and Spearman correlation coefficients for each pair of traits in ‘MLH\_output.csv’. *P*-values for each correlation as well as *P*-values adjusted for multiple comparisons using the Benjamini-Hochberg correction are also listed in separate columns.
2. Additionally, a pdf called, ‘corr\_chart\_pearson.pdf’ is generated, which creates a matrix where the diagonal cells contain plots of the distribution corresponding to each column in MLH\_output.csv, the cells (i,j) in the lower triangle of the matrix depict scatterplots using the ith and jth diagonals as x and y values, respectively, and the cells (i,j) in the upper triangle of the matrix depict Pearson correlation coefficients between the ith and jth traits with (uncorrected) significance depicted as red dots and stars, such that “\*\*\*\*”, “\*\*\*”, “\*\*”, “\*”, and “.” correspond to *P*-values of 0, 0.001, 0.01, 0.05, and 0.1, respectively. Another pdf file called, ‘corr\_chart\_spearman.pdf’ is generated with the same properties excepting Spearman correlation coefficients replace Pearson correlation coefficients in the upper triangle of the matrix.
3. In addition to listing the *P*-values adjusted for multiple comparisons in ‘all\_correlations.csv’, the pipeline creates a file called, ‘fdr\_cutoffs.txt’ that reports the unadjusted *P*-value representing the new significance threshold after correction for multiple comparisons for both the Pearson and Spearman correlations. Assessing significance based on these thresholds will retain the same hypothesis tests that are retained by the algorithm that actually transforms the *P*-values themselves.
4. The last four items that are generated by the pipeline in the ‘Correlations’ directory are csv files that largely match the format of the pdf files described in Step 2: they contain matrices with the trait names in the diagonals, and the correlation coefficients are listed in the upper triangle, but the lower triangle contains *P*-values corresponding to the correlation test between the ith and jth traits. The four files correspond to Spearman and Pearson correlation coefficients, both with and without adjusting the *P*-values for multiple comparisons using the Benjamini-Hochberg method (5% FRD), and each file is labeled accordingly (‘correlation\_chart\_pearson .csv’, ‘correlation\_chart\_pearson\_adjusted.csv’, ‘correlation\_chart\_spearman.csv’, and ‘correlation\_chart\_spearman\_adjusted.csv’).

## Testing for Single-Locus Associations (SLAs) with an F-ratio Test

1. At this point, you are prompted to test the dataset for single locus effects by comparing a model invoking the heterozygosity of each locus as an independent predictor of a particular trait to a model invoking only the multi-locus heterozygosity estimator as a response variable (Szulkin et al., 2010). If you opt to perform this analysis by typing, “y” or “yes”, you are prompted to list each trait for which you would like to perform an F-ratio test (the trait options are listed after you type “y” and hit enter). Type as many of these as desired (pressing enter between them), and then type, “done”.
2. The pipeline then prompts the user to enter the multi-locus heterozygosity estimator to be used for the model comparison test. MLH is recommended, but the user can also use IR, SH, or HL. Type whichever estimator desired, and press enter.
3. At this point, you are prompted to designate an error distribution and link function for each trait being used as the response variable in the model comparisons. The error distribution and link function designation is used in generalized linear models to relate the response variable (which is allowed to be non-normally distributed and have non-constant variance) to the predictors. If the response variable is normally-distributed with constant variance, the error distribution is the default “gaussian” distribution (with a default “identity” link function), and you would type, “guassian”. If the response variable is binary, the error distribution is “binomial” (with a default “logit” link function), and you would type, “binomial”. The GLM() function in R, which performs the regression analyses at this step in the pipeline, can accommodate other families of error distributions and link functions for more complex response variables as well (e.g., “poisson”). The pipeline prompts the user to designate an error distribution separately for each trait that the user entered (it lists each trait before it prompts the user).
4. The module described in steps 1-3 generates four types of output files.
   1. The first is a file called, ‘fitness\_and\_loci\_combined.csv’, that combines all of the data from ‘MLH\_output.csv’ with all of the data from ‘heterozygotes\_and\_homozygotes.csv’ except that “NA”s in the single-locus columns are replaced by the mean heterozygosity at that locus for the samples that *were* typed (see (David et al., 2007) for details).
   2. The second type of output is generated for eachtrait that the user entered in Step 1 and is a text file called, ‘Multiple\_regression\_vs\_single\_regression\_[trait name here] .txt’ that contains all of the information regarding significant predictors for each of the two models described in Step 1 as well as the F-ratio test comparing the two models (if this result is significant, it suggests that the two models are different and provides support for a hypothesis positing the existence of a single-locus association for the trait in question)—the significance of this F-ratio test is displayed as the last line of each text file.
   3. The third type of output is again generated for each traits that the user entered in Step 1; it is a tiff file called, ‘Multiple\_regression\_[trait name here].tiff’ that contains plots that can help the user decide whether the assumptions of linear regression are satisfied for the model parameterized by each locus treated as a separate predictor. Note that interpreting residual plots for models invoking non-Gaussian error distributions is not always appropriate.
   4. The fourth type of output is again generated for each traits that the user entered in Step 1; it is a tiff file called, ‘Single\_regression\_[trait name here].tiff’ that contains plots that can help the user decide whether the assumptions of linear regression are satisfied for the model parameterized only by the single multi-locus heterozygosity predictor. Note again that interpreting residual plots for models invoking non-Gaussian link functions is not always appropriate.

## Running Regression Analyses

In this portion of the pipeline, you can regress any trait on any other trait or combination of traits (including the addition of interaction terms). Less broadly, this portion of the pipeline is intended to act as the “heterozygosity-fitness correlation(s)” (HFCs are actually regressions).

The module is divided into three components: full model regressions, data subset regressions, and user-generated data subset regressions. The pipeline asks whether you want to run regression analyses, and will allow you to opt in or out of each of the three components in order.

### Full Model Regressions

1. In the full model, you are provided with a list of fitness variables in the dataset. Before typing the response variable desired for this particular model, you are prompted to type the error distribution associated with the distribution of the response variable (see Step 3 in the Testing for Single-Locus Associations (SLAs) with an F-ratio Test section for details about the error distribution).
2. You are then prompted to type the response variable. Type the name and press enter (the pipeline will throw an error is you type a name that does not exist in the data set).
3. You are then prompted to enter predictor variables, and you are free to type as many of these as are relevant (MLH being the most relevant for HFC studies), pressing enter between each entry. The pipeline accommodates interaction terms here as well (e.g., MLH\*PC1\_cov in the example data set). Type “done” or “Done”.
4. At this point, the module generates two files in the ‘Regressions’ subdirectory in the pipeline directory: 1) ‘[response variable name here]on[predictor variable names here]\_fullModel.txt’ which lists regression statistics exactly as if the GLM() (generalized linear model) function had been called in *R* and 2) ‘[response variable name here]on[response variable names here].txt’ which applies the Akaike Information Criteria to the full model, and lists which predictor variables are retained in the final model, exactly as if the step() function had been called in *R*. It is important to bear in mind for the interpretation of these regression results that models using non-Gaussian error distributions undergo transformations such that the regression coefficients will not have a direct biological interpretation.
5. You are asked whether you want to run another model, and upon selection of “y”, Steps 1-4 are repeated (i.e., you are allowed to select other response and/or predictor variables in a new model). If “n” is entered, the module moves on to Data Subset Regressions.

### Data Subset Regressions

The Data Subset Regressions component acts exactly as the Full Model Regressions component does, with only a few changes.

1. Upon typing “y” and pressing enter when prompted to run HFCs on only certain individuals bearing certain trait values, you are provided with a list of trait names and some sample trait values for those traits. At this point, the user types the trait and trait value, separated by a comma and *no space*, that together specify the subset of data that is desired.
   1. It is possible to create subsets of data using permutations of traits and values across several different traits or even within a single trait (provided that this trait has more than two states). For instance, in the example data provided, let’s say we want to create a dataset that only contains individuals that have been infected by two different viruses, V1, and V2. In that case, the user would type, “V1,0”, press enter, and then type, “V2,0”, specifying that samples negative for V1 and samples negative for V2 should be excluded.
   2. Type, “done” when you are done with the specifications.
2. The pipeline’s prompts will then look exactly like those described in the Full Model Regressions section, but the results will be saved in a new nested subdirectory, ‘Data\_subsets’, inside the pipeline directory, with another subdirectory inside ‘Data\_subsets’ named after the trait-by-trait value combination(s) specified by the user (‘V1\_0\_V2\_0\_removed\_subset’ in our example case).
3. Inside this specific subdirectory, the pipeline generates a file containing only the individuals not bearing the trait values specified by the user (called ‘V1\_0\_V2\_0\_removed\_subset.csv’ in our example case), and the regressions the user then generates are based on this dataset.
4. The new regression results are placed into a ‘Regressions’ directory as described in the Full Model Regressions section; the new ‘Regressions’ directory is nested within this specific subdirectory.
5. Continue looping through this component (following prompts to do so) for as many regression models and data subsets as needed.

### User-Generated Data Subset Regressions

If you have another data set(s) that you want to analyze that does not, for whatever reason, lend itself well to being subdivided based on specific trait values, the third component of the HFC module can accommodate this.

1. Upon responding “y” and pressing enter to the prompt, “Do you want to run regressions on csv files of your own making?”, you are asked to specify the path of the csv file you intend to use. Let’s say, for example, that we wish to analyze data in the example dataset that are infected by any permutation of viruses V1, V2, and V3. This is relatively tough to accomplish using the protocol described in the Data Subset Regressions section, but relatively straightforward to accomplish manually in a spreadsheet.
2. After creating the csv file manually, specify the file’s path (using the example data provided, I would type, ‘/Users/mf/Desktop/pipeline/ At\_least\_1\_virus\_infection.csv’) .
3. You are then prompted to type a directory name in which to store the results (e.g., ‘At\_least\_1\_virus\_infection’) and press enter. The rest of this component of the module behaves as in the Full Model Regressions section.
4. Continue looping through this component (following prompts to do so) for as many regression models and user-generated data subsets as needed.

## Converting Regression Results into Spreadsheets

The last thing *HeFPipe* does is provide the user with the option to convert regression results generated from any permutation of the regression components (Full Model Regressions, Data Subset Regressions, or User-Generated Data Subset Regressions) into csv files.

1. In each case, the pipeline generates a new directory called ‘processed\_model\_output’, which is nested in the ‘Regressions’ directory created by any of the three components (recall that either subset component will generate a ‘Regressions’ directory nested within the subset-specific directories).
2. The pipeline will scan the relevant ‘Regressions’ directory for ‘.txt’ files and will use regular expressions to extract the trait name, regression coefficient, standard error, score of the statistical test, *P*-value of regression test, and type of score (e.g., t- or Z-score) from the text file, and will list these in a csv-formatted file called, ‘[error\_distribution\_designation here][response\_variable\_here]on[predictor\_variable(s)\_here] \_fullModel\_processed.csv’ (e.g., ‘gaussianQnWeight\_initialonMLH\_fullModel\_processed’)
3. In the case of Data Subset Regressions or User-Generated Data Subset Regressions, the pipeline will ask you to specify the path of the ‘Regressions’ directory to use. For instance, in the example data provided, we would type (or drag the directory into terminal) “/Users/mf/*HeFPipe*\_repos/Example\_data/2\_After\_the\_pipeline\_is\_run/pipeline/Data\_subsets/V1\_0\_V2\_0\_removed\_subset/Regressions/ “ or “/Users/mf/*HeFPipe*\_repos/Example\_data/2\_After\_the\_pipeline\_is\_run/pipeline/At\_least\_1\_virus\_infection/Regressions/” for the Data Subset Regressions or User-Generated Data Subset Regressions examples, respectively.
4. In the case of Data Subset Regressions or User-Generated Data Subset Regressions, repeat Step 3 for as many data subsets as desired and/or generated.

# References

Alho, J. S., Välimäki, K., Merilä, J., Valimaki, K., & Merila, J. (2010). *Rhh*: an R extension for estimating multilocus heterozygosity and heterozygosity-heterozygosity correlation. *Molecular ecology resources*, *10*(4), 720–2. doi:10.1111/j.1755-0998.2010.02830.x

Amos, W., & Acevedo-Whitehouse, K. (2009). A new test for genotype-fitness associations reveals a single microsatellite allele that strongly predicts the nature of tuberculosis infections in wild boar. *Molecular Ecology Resources*, *9*(4), 1102–1111. doi:10.1111/j.1755-0998.2009.02560.x

Balloux, F., Amos, W., & Coulson, T. (2004). Does heterozygosity estimate inbreeding in real populations? *Molecular ecology*, *13*(10), 3021–31. doi:10.1111/j.1365-294X.2004.02318.x

David, P., Pujol, B., Viard, F., Castella, V., & Goudet, J. (2007). Reliable selfing rate estimates from imperfect population genetic data. *Molecular ecology*, *16*(12), 2474–87. doi:10.1111/j.1365-294X.2007.03330.x

Frankham, R., Ballou, J. D., & Briscoe, D. A. (2010). *Introduction to conservation genetics* (2nd ed.). New York: Cambridge University Press.

Raymond, M., & Rousset, F. (1995). *GENEPOP* (Version 1.2): Population Genetics Sofware for Exact Tests and Ecumenicism. *Heredity*, *86*, 248–249.

Slate, J., & Pemberton, J. (2006). Does reduced heterozygosity depress sperm quality in wild rabbits (Oryctolagus cuniculus)? *Current Biology*, *16*(18), 790–791. Retrieved from citeulike-article-id:7912871

Szulkin, M., Bierne, N., & David, P. (2010). Heterozygosity-fitness correlations: a time for reappraisal. *Evolution*, *64*(5), 1–16. doi:10.1111/j.1558-5646.2010.00966.x