# User manual for `LASSI`

Alexandre M. Harris and Michael DeGiorgio

June 20, 2019

## Contents

**1 Introduction** **2**

**2 Operation** **2**

2.1 Running `LASSI` from the command line . . . 3

2.1.1 Initial scan of the data to generate frequency spectra and count spectra . . . 3

2.1.2 Obtaining the mean neutral spectrum . . . 5

2.1.3 Running the final scan to compute the $T$ statistic . . . 5

2.2 Converting VCF files to `LASSI`-compatible format with `vcf2ssx` . . . 6

**3 Default input file format** **7**

**4 Output format** **8**

**5 Examples** **9**

## 1 Introduction

Welcome to `LASSI`, our **L**ikelihood-based **A**pproach for **S**elective **S**weep **I**nference. `LASSI` is a Python program for identifying selective sweeps, as well as classifying them as hard or soft, from multilocus polymorphism data. Input data for your study population can be either phased as haplotypes or unphased as multilocus genotypes (MLGs), and `LASSI` makes no distinction between these as long as the proper formatting is followed (see Section 3:*Default input file format*). Selective sweeps are detected from an elevated value of the $T$-statistic, which is the log composite likelihood ratio of a sweep model compared to the genome-wide background (taken as a model of neutrality), and indicates the likelihood that a genomic region fits a sweep model. For regions with elevated values of $T$, sweeps are inferred to be hard or soft from the model parameter $m$. See Harris and DeGiorgio [2019b] for a complete explanation of the theory underlying `LASSI`.

Note that you will likely need to convert the format of your data for compatibility with `LASSI`. For simplicity between platforms, we reuse the input format of our previous software, `SS-X12` (see Harris and DeGiorgio [2019a] for a description of this method). To assist in converting your data, we provide the Python script `vcf2ssx`, which converts VCF files to a format compatible with both `LASSI` and `SS-X12`. Thus, if you have previously used `SS-X12` and converted your VCF files with `vcf2ssx`, then you are already set to begin analysis with `LASSI`. We describe the operation of `vcf2ssx` in Section 2.2: *Converting VCF files to* `LASSI`*-compatible format with* `vcf2ssx`, and provide examples of properly-formatted files in Section 5:*Examples.*

Please contact Alex Harris (`amh522@psu.edu`) if you encounter any issues in your use of these programs. As necessary, please also include an example converted data file (such as the first 1000 SNPs of your format-converted data).

If you use either `LASSI` or `vcf2ssx` software, then please cite it as

> A M Harris and M DeGiorgio (2019). A likelihood approach for uncovering selective sweep signatures from haplotype data. *bioRxiv.*

## 2 Operation

`LASSI` and `vcf2ssx` are meant for use on UNIX systems and are formatted for Python 2.7 specifically. We do not guarantee compatibility with Python 3+. We do not provide support for errors arising due to system incompatibilities, but if you do encounter an error, then please first check your invoked version of Python. We distribute our software in compressed (.tar.gz) format, and include the `LASSI` pipeline (five Python scripts in total), `vcf2ssx`, this manual, and example data with output, as part of your download. To unpack the `LASSI` program directory from the command line, enter

```
tar -xzvf LASSI_program.tar.gz

cd ./LASSI_program
```

These commands will create the directory `LASSI_program` within the current directory, and switch the user to `LASSI_program`. For simplicity, we recommend that all input dataset files be in the same directory as the `LASSI` scripts, but this is not required. Additionally, output files are written to the local directory.

## 2.1 Running `LASSI` from the command line

The operation of `LASSI` to identify and classify selective sweeps is performed in several steps as part of a pipeline, though users will only need to invoke the central script, `LASSI_iterator.py`, which itself calls the other four scripts. The pipeline is composed of three basic steps, which we outline here and describe in detail in the next sections (2.1.1-2.1.3).

First, the sample is scanned via sliding window to extract the haplotype (or MLG) frequency spectrum at each genomic position to be analyzed (run option `initial`, Section 2.1.1). The counts for each haplotype (or MLG) are also recorded. The number of genomic positions in the analysis is a function of the window size and the distance by which the window advances with each step (the step size). Whole and truncated spectra of frequencies and counts are produced in each scan, comprising four spectra in all. Whole spectra contain the frequencies of all classes within the analysis window, whereas truncated spectra are cut off at a user-specified number $K$ of most-frequent haplotypes. We have generally found that a truncation of $K = 20$ provides accurate results when detecting selective sweeps.

Following the initial scan, a mean neutral background spectrum is generated from the haplotype (or MLG) frequency spectra of all the analysis windows, providing our null model. This spectrum is required to compute the likelihood function that quantifies candidate selective sweeps.

Finally, using the neutral spectrum for the dataset and the spectrum of counts for each analysis window obtained during the initial scan, a $T$-statistic is assigned for each window in the final step. Also outputted for each window is its genomic coordinate (central SNP in the window), the inferred number of sweeping haplotypes $m$, the value of $\varepsilon$ used in the model, and the value of the smallest haplotype class in the neutral spectrum $p_K$ (see Harris and DeGiorgio [2019b], *Theory*).

All scripts provided with the software download (`LASSI_iterator.py`, `LASSI_spectrum_and_Kspectrum.py`, `LASSI_Kspectrum_ONLY.py`, `LASSI_average_Kspect.py`, and `LASSI_scan.py`) must be present within your working directory for the `LASSI` pipeline to function properly. Additionally, we recommend that unrelated output files be moved to separate destination directories to prevent new results from appending to existing files across analyses.

### 2.1.1 Initial scan of the data to generate frequency spectra and count spectra

Generating the $T$ statistic to identify selective sweeps fundamentally depends on obtaining and leveraging frequency and count spectra for the dataset under analysis. To do this, use the `initial` option as the first argument for `LASSI_iterator.py` and wait for the `All done` message to appear on the command line. `LASSI` will also report the size of the final analysis window, which is unlikely to contain the desired number of SNPs (users can then decide whether to use this window in their analysis). The run time of the initial scan depends on the choice of window and step sizes. Choosing smaller windows will result in faster per-window computations, while choosing smaller step sizes will result in a greater overall number of computations performed.

The `initial` step will output spectra—full counts, truncated counts, full frequency, and truncated frequency—for each analysis window to chromosome-specific `.txt` files, as well as an `allreps_` file (also `.txt`) that contains all of the truncated frequency spectra across all chromosomes and a `window_centers_` file (`.txt`). The `allreps_` file will be used in the next step (2.1.2) to generate the neutral background spectrum (the `total_neutral_` file), while the `window_centers_` file allows `LASSI` to assign a genomic coordinate to each $T$ statistic. To proceed, enter the following into the command line:

```
python LASSI_iterator.py initial <chromosome_number> <truncation>
<unique_ID> <file_name_components> <study_population> <headfile>
<window_size> <step_size> <to_allreps>
```

There may be instances in which new truncations for spectra are desired. For such situations, we provide an alternate script to the one above, which takes the existing whole spectra from a previous scan (such as the example scan invoked above) and truncates them differently. This does not change the window or step sizes, allowing for reuse of the `window_centers_` file. To create a new set of truncated spectra, invoke:

```
python LASSI_iterator.py rescan <chromosome_number> <truncation>
<unique_ID> <file_name_components> <study_population> <headfile>
<window_size> <step_size> <to_allreps>
```

Ten arguments in total are specified for the above commands, including the command `option` (`initial` or `rescan`) itself:

1. `option`, which is constrained to be either `initial` or `rescan`; no other choice will work for generating spectra, and `rescan` can only be used on spectra previously generated with `initial`
2. `chromosome_number`, integer or string identifying the contig of your data to be scanned (such as 1, X, 2L, etc.)
3. `truncation`, positive integer indicating the number of classes to be included in truncated spectra; for example, choosing 20 means that only the top 20 frequencies/counts will be included (these will be weighted values)
4. `unique_ID`, a string that defines what the output file will be called, such as `my_scan_122519`; output prints to local directory into a `.txt` file
5. `file_name_components`, comma-separated strings comprising the name of the input file before and after the contig identification; this makes it convenient to parallelize scans when a dataset consists of, for example, multiple chromosomes whose data are stored within similarly named files; format this as `file_name_before_ID,file_name_after_ID`
6. `study_population`, string indicating which population to scan, formatted exactly as in the `headfile`

7. `headfile`, the name of the file containing the population assignment for each individual in the input file

8. `window_size`, positive integer corresponding to the number of SNPs to be included within the analysis window

9. `step_size`, positive integer corresponding to the step size, in SNPs, that the analysis window moves along the chromosome after each computation; typically 1/10 the `window_size`

10. `to_allreps`, either `yes` or `no`, indicating whether the truncated frequency spectrum of each window should additionally output to an `allreps_` file which will be used in the next step to create the neutral background frequency spectrum (`total_neutral_` file); this should generally be `yes` because the neutral background frequency spectrum needs to be computed for each parameter set

### 2.1.2 Obtaining the mean neutral spectrum

Once the `allreps_` file containing the truncated frequency spectra for each window has been generated across all chromosomes, it is used to generate a neutral background spectrum (`total_neutral_` file), which is the mean spectrum for the entire chromosome. To obtain a master spectrum for the entire genome across chromosomes, you will need to take the weighted average of each `total_neutral_` spectrum generated for each chromosome.

Using `neutavg` as the `option` argument to generate a `total_neutral_` file is straightforward, retaining five arguments of `initial` and `rescan`, with one subsequent argument:

```
python LASSI_iterator.py neutavg <chromosome_number> <truncation>
<unique_ID><file_name_components> <study_population> <allreps_lines>
```

The `allreps_lines` argument is an integer indicating the total number of lines in the `allreps_` file. You will need to use the `wc -l` command from the terminal for your UNIX system to count the number of lines in the `allreps_` file. The number of lines of an `allreps_` file is equivalent to the total number of windows scanned across all chromosomes.

### 2.1.3 Running the final scan to compute the $T$ statistic

The final step in the `LASSI` pipeline is to assign a $T$ statistic (as well as the number of sweeping haplotypes/MLGs $\widehat{m}$, and non-sweeping haplotype parameter $\widehat{\varepsilon}$) to each analysis window defined in the `initial` scan or `rescan`. This is achieved by computing the likelihood of neutrality, using the neutral background spectrum, and the likelihood of a sweep, using a distortion of the neutral background spectrum. The likelihood computation is the most time-intensive component of the pipeline, as it requires optimization over both $m$ and $\varepsilon$. This step is performed with `MLcalc` as the `option`, and once again five arguments in common with the previous steps:

```
python LASSI_iterator.py MLcalc <chromosome_number> <truncation>
<unique_ID> <file_name_components> <study_population> <out_option>
```

The argument `out_option` specifies the sweep model to be used in the scan, and is an integer between 1 and 5. The difference between models, however slight, is their stringency in assigning a candidate sweep as hard ($\widehat{m} = 1$) or soft ($\widehat{m} > 1$). Options 1 through 4 use a sweep distortion model that unequally adds weight to the sweeping haplotype classes, whereas option 5 adds weight equally to each. The options represent:

1. A model in which weight is added to sweeping classes proportional to $1/i$, where $i$ is the haplotype class
2. A model in which weight is added to sweeping classes proportional to $1/i^2$
3. A model in which weight is added to sweeping classes proportional to $e^{-i}$
4. A model in which weight is added to sweeping classes proportional to $e^{-i^2}$
5. A model in which weight is added to sweeping classes proportional to $1/m$, where $m$ is the number of sweeping haplotype classes in the model

Options 1-4 are increasingly more likely to classify sweeps as soft, whereas option 5 is the least likely to classify a sweep as soft. We have found that option 3 provides the greatest detective power and most accurate classifications of sweeps in simulation experiments, though the difference across all models is marginal. See Harris and DeGiorgio [2019b] for a more complete description of this theory.

## 2.2 Converting VCF files to `LASSI`-compatible format with `vcf2ssx`

**We have reproduced this section of the* `LASSI` *manual from the equivalent section of our* `SS-X12` *manual.**

Here, we describe the operation of `vcf2ssx`, a Python script which converts VCF files to compressed files in `LASSI` format (as .txt.gz). `LASSI` is compatible with either compressed (must contain ".gz" in the file name) or uncompressed input data files. Invoke `vcf2ssx` with the command

```
python vcf2ssx.py <vcf_infile> <formatting> <nonbi_removed>
[<chromosome_index>] [<position_index>] [<rsID_index>] [<refAllele_index>]
[<altAllele_index>] [<data_index>]
```

A minimum of three arguments is required, with six additional arguments depending on the choice of `formatting`:

1. `vcf_infile`, the name of the VCF file to be converted to `LASSI` format; for example `my_data.vcf.gz` (note that uncompressed VCFs are also compatible with `vcf2ssx`)
2. `formatting`, defines whether the order of columns in the VCF follows the standard VCFv4.2 format or not; answering `yes` indicates formatting following VCFv4.2, meaning that `vcf2ssx` requires no further arguments; answering `no` subsequently requires the user to define the index positions of important columns in the infile (see items 4-9)

3. `nonbi_removed`, defines whether the user has removed all sites from their input VCF file that are NOT biallelic SNPs; answering `yes` indicates that such sites are removed, while answering `no` indicates that such sites are retained in the input (requiring `vcf2ssx` to filter these out)
4. `chromosome_index`, optional argument if `formatting` is `no`, integer specifying the index of the chromosome number column in the VCF input file (indexing begins from zero for this and subsequent arguments)
5. `position_index`, optional argument if `formatting` is `no`, integer specifying the index of the physical position column in the VCF input file
6. `rsID_index`, optional argument if `formatting` is `no`, integer specifying the index of the SNP rsID column in the VCF input file
7. `refAllele_index`, optional argument if `formatting` is `no`, integer specifying the index of the reference allele column in the VCF input file
8. `altAllele_index`, optional argument if `formatting` is `no`, integer specifying the index of the alternate allele column in the VCF input file
9. `data_index`, optional argument if `formatting` is `no`, integer specifying the first data column's index in the VCF input file

A complete command line invocation of `vcf2ssx`, with standard formatting of the VCF input file and sites that are not biallelic SNPs removed, is:

```
python vcf2ssx.py my_data.vcf.gz yes yes
```

## 3 Default input file format

**We have reproduced this section of the* `LASSI` *manual from the equivalent section of our* `SS-X12` *manual.**

Here, we discuss the default format of the data file which `LASSI` takes as input, as well as the `headfile` that must accompany the input data. The example data files included within the `LASSI` directory are formatted according to our requirements.

The `LASSI` input data file is individual-delimited and contains data for all individuals from all study populations, as only one data file is analyzed per run. Additionally, each input data file must contain data from only one chromosome. Be sure to include `_hap` at the end of the filename to indicate phased haplotype data, or `_mlg` to indicate unphased MLG data. We recommend running analyses on multiple chromosomes in parallel with a multithreading or multiprocessing module (such as `parallel` in R or `multiprocessing` in Python).

For $S$ analyzed SNPs and $n$ sampled diploid individuals, the input data file contains $S$ lines of data. Each line consists of summary information on the SNP—chromosome ID, SNP rsID, physical position, reference allele, and alternate allele—followed by $n$ entries corresponding to the allelic state of each sampled individual at that SNP. Thus, data lines contain $n + 5$ entries. Each data line should look as follows:

12 rs536857393 93597 C T 1 1 3 1 1 4 4 2 4 2

For phased haplotype data, an individual's allelic state is coded as 1, 2, 3, or 4. States 1 and 4 correspond to homozygous genotypes for the reference (VCF format: 0|0) and alternate alleles (1|1), respectively. States 2 and 3 correspond to heterozygous genotypes, where the alternate allele is located on the second haplotype in state 2 (0|1) and on the first haplotype in state 3 (1|0). For unphased diploid MLGs, states 2 and 3 are not included. Instead, an individual's heterozygous allelic state is 5 (1/0 or 0/1). Missing sites are encoded as the letter "N", corresponding to each instance of "." in the VCF.

The `headfile` must contain a single line of $n$ entries. Each entry is the name of a sampled population, and this name must be invoked exactly within the `study_population` argument (see above). Population assignments are indexed to match the position of individuals in the input data file, such that an entry at position $p$ in the `population header` file is the population assignment of the individual at position $p + 5$ in the input data file. Note that only one population at a time can be scanned with `LASSI`. The header line should look as follows (two populations, CEU and CHB are included here; same example as above):

CEU CEU CEU CEU CEU CHB CHB CHB CHB CHB

Although our example here features only 10 individuals, with 5 from either population, we caution that `LASSI` performs better for larger sample sizes. That is, the sample size needs to be sufficient for the captured variation to properly distinguish between neutral and selected regions of the genome. To make this determination, we recommend running `LASSI` on a few megabases (Mb) of your data and observing the contrast in value between signal peaks and signal valleys. We also find that we have more power to detect sweeps from phased haplotype data than from MLG data (for the same number of individuals, the MLG sample size is half of the haploid sample size). Analyses in Harris and DeGiorgio [2019b] feature samples of 100 diploids per population.

## 4 Output format

The `LASSI` pipeline yields various output files (uncompressed, plain text, `.txt`) as the analysis progresses, and these can be divided into three categories: spectra, window centers, and sweep scan files. Files are of length equal to the number of analysis windows in the data file. We will devote the most attention to the format of sweep scan files due to their somewhat less intuitive structure, though we emphasize that all output files are easy to parse.

Spectrum files can be either frequencies or counts, as well as whole or truncated. Thus, each chromosome that is analyzed has four associated spectra (not counting the `allreps_` file, which covers all chromosomes). Spectra are arranged in decreasing size of class, such that the largest class is first, and smallest is last. The same line across each file corresponds to the same analysis window, meaning that, for example, the 150th line in the truncated frequency spectrum derives from the same analysis window as the 150th line in the whole counts spectrum. The sum of all classes for each line (window) of the frequency spectrum files is 1, and the sum of all classes for each line of the counts spectrum files is equal to the diploid sample size for the population. For truncated spectra, the values of each class are weighted to ensure correct values. For the `initial` run of `LASSI`, a `window_centers_` file is generated along with the four spectra that also corresponds to them, line-for-line, indicating the nucleotide coordinate of the center of the analysis window, as defined by the input file (the coordinate is the only element of each line).

Each output line of the sweep scan file generated with `MLcalc` is space-separated, with six elements per line. Once again, each line in the output derives from the same analysis window as in the spectrum files. A `LASSI` output line consists of the following:

- $T$ statistic, a measure of the support for a selective sweep at the analysis window, with larger values indicating greater support for a sweep
- $\widehat{m}$, the inferred number of sweeping classes in the population, based on the optimal sweep model
- $\widehat{\varepsilon}$, the most optimal inferred frequency of the least frequent class
- $p_K$, the value of the smallest frequency class in the neutral background spectrum; the combination of $p_K$ and $\varepsilon$ defines the values of the non-sweeping frequency classes under the sweep model; this is set by default to be the value of the largest non-sweeping frequency class
- Likelihood model choice, integer numbered 1-5, corresponding to the selections enumerated in Section 2.1.3
- Coordinate of analysis window center, in nucleotides

## 5 Examples

In the `examples` directory within `LASSI`, we provide compressed example data in both `LASSI` format and as VCF, as well as example output files. Data come from Phase 3 of the 1000 Genomes Project [Auton et al., 2015]. The included data consist of the `headfile` for the 2,504 sequenced individuals within the 1000 Genomes Project dataset, the first 1000 SNPs of chromosome 12 from the 1000 Genomes Project (as `.vcf.gz`, `.txt.gz`, and `.txt`), and the output files for each stage of the `LASSI` pipeline from the scan of `chr12_first1000_hap.txt.gz`. Each stage of output has been manually placed in a directory named according to the `option`. We analyzed the YRI population for these examples.

The commands required to generate the example output files are:

- `python vcf2ssx.py chr12_first1000_hap.vcf.gz yes yes`
  This converts the `.vcf.gz` file to a `.txt.gz` file; note that the VCF file is not required to have `_hap` or `_mlg` in the filename at this stage.
- `python LASSI_iterator.py initial 12 15 LASSI_EXAMPLE chr,_first1000_hap.txt.gz CEU 1000genomes_phase3_head.txt 117 12 yes`
  Command to generate the spectra for CEU chromosome 12, using a truncation of $K = 15$,

unique ID of `LASSI_EXAMPLE`, window size of 117 SNPs, step size of 12 SNPs, and truncated frequency spectrum output additionally redirected to an `allreps_` file. Output consists of six files: four standard spectra (`_count_Uhap`, `_count_Khap`, `_spectrum_Uhap`, `_spectrum_Khap`), the `allreps_` file (identical to `_spectrum_Khap` for this example due to the absence of other parallel scans), and a `window_centers_` file.

- `python LASSI_iterator.py rescan 12 20 LASSI_EXAMPLE chr,_first1000_hap.txt.gz CEU 1000genomes_phase3_head.txt 117 12 yes`
  Performs a truncation of `_count_Uhap` and `_spectrum_Uhap` from the previous command to output new truncated files ($K = 20$) and new `allreps_` file.

- `python LASSI_iterator.py neutavg 12 20 LASSI_EXAMPLE chr,_first1000_hap.txt.gz CEU 3`
  This command reads in the $K = 20$ `allreps_` file from the previous command and takes the average of each haplotype frequency class across all windows to generate the neutral background frequency spectrum, `total_neutral_` (note once again that you will need to determine the number of lines within your `allreps_` file and use this as the final argument).

- `python LASSI_iterator.py MLcalc 12 20 LASSI_EXAMPLE chr,_first1000_hap.txt.gz CEU 3`
  Using the `total_neutral_` and `_count_Khap` files, a $T$ statistic is assigned to each window of the $K = 20$ scan, using likelihood model 3. The single output file is called `LASSI_EXAMPLE_K20_chr1hap.txt`

## References

A Auton, G R Abecasis, and The 1000 Genomes Project Consortium. A global reference for human genetic variation. *Nature*, 526:68–74, 2015.

A M Harris and M DeGiorgio. Identifying and classifying shared selective sweeps from multilocus data. *bioRxiv*, 2019a.

A M Harris and M DeGiorgio. A likelihood approach for uncovering selective sweep signatures from haplotype data. *bioRxiv*, 2019b.