A Legofit Tutorial

Alan R. Rogers

July 7, 2022

1 Introduction

This is a tutorial on Legofit, a package that uses genetic data to estimate the history of population size, subdivision, and admixture [4, 5]. It will step you through the process of installing the software, using it to tabulate the frequencies of nucleotide site patterns, and then analyzing these data to fit a model of population history. We will assume that data files are already available in .vcf format. The full Legofit documentation is available online.

This tutorial will be used in the 2022 edition of the AGAR Workshop. During the workshop, we will do only last step in this process, which estimates population history from site pattern frequencies. We will not do the preliminary steps, which tabulate these frequencies. Nonetheless, this tutorial describes the whole process.

We plan to introduce students to the Center for High-Performance Computing (CHPC) at the University of Utah, and some of the sections below assume that you have an account on that system. (We'll be getting accounts for everyone enrolled in the workshop.) These sections involve using the SLURM Workload Manager to submit jobs to the compute cluster. However, our focus during the workshop will be on the last step, which can be done in two ways: either on the CHPC (using SLURM), or on your own computer (using the bash shell). This tutorial will cover both environments.

The material below is organized into the following sections:

Try to install Legofit (sec. 2) on your own before the workshop starts. We'll help with installation problems during the workshop, but that will take less time if you've already made a start. The workshop will not cover secs. 4–5. Those sections are only for reference, when you work with Legofit on your own. During the workshop, most of our efforts will be on fitting models (sec. 6), but we'll spend some time at the end on model selection, model averaging, and graphing (secs. 7–8).

2 Installing Legofit

This section explains how to get Legofit running on your own computer. It's already installed on the CHPC, so these steps are not needed there.

2.1 Install git if necessary

You may have git already. To find out whether you do, type:

```
type git
```

On Mac or Linux, this will print the location of the executable git file. If there is no such file, you'll get an error message saying `type: git: not found`. I recommend using homebrew to install packages on a Mac. To install git using homebrew, type:

```
brew install git
```

Otherwise, visit git's website and look for the download instructions.

2.2 Clone legofit

Use git to clone Legofit onto your machine. This step will create a subdirectory called "legofit," which will contain the source code. Before executing the command below, use the `cd` command to move into the directory where you keep source code that other people have written. I keep such code in a directory called `distrib`, which I originally created by typing `mkdir distrib`. Having created such a directory, move into it by typing

```
cd distrib
```

Then clone legofit by typing:

```
git clone https://github.com/alanrogers/legofit.git legofit
```

This will create a directory called legofit. Use `cd` to move into it.

2.3 Install a C compiler if necessary

To see whether you have a C compiler already, type `type cc`, or `type gcc`, or `type clang`. You should get output like

```
cc is /usr/bin/cc
```

If you need to install a C compiler, there are several alternatives. On a Mac, you can install Xcode from the App Store. Or you can use Homebrew to install clang or gcc:

```
brew install clang
```

or

```
brew install gcc
```

On Linux, the command would be

```
sudo apt-get install clang
```

or

```
sudo apt-get install gcc
```

2.4 Install “make” if necessary

You will also need the program “make,” so type “type make.” If you need to install, then

```
brew install make
```

2.5 Set up a “bin” directory to hold executable files.

On Unix-like operating systems (MacOS and Linux) it is conventional for each user to maintain a directory named ”bin”, just below the home directory, which contains executable files. This directory must also be added to the system’s PATH variable, which is used for finding executable files.

First create a “bin” directory, if you don’t already have one. To do so, first type the command

```
cd
```

at the shell prompt. This moves you into your home directory. Then type

```
mkdir bin
```

This creates a new directory called “bin.”

You now need to add this to your shell’s PATH variable. I’ll assume you’re using the bash shell. In your home directory, look for a file called either “`.bash_profile`” or “`.profile`”. Because the name begins with “.”, it will not show up if you type `ls`. However, it will appear if you type `ls .bash_profile` or `ls .profile`. If only one of these files exists, open it with a text editor (not a word processor). If they both exist, edit “`.bash_profile`”. If neither exists, use the editor to create a new file called “`.bash_profile`”.

Within this file, you may find existing definitions of the PATH variable. Add the following after the last line that changes this variable:

```
export PATH=$HOME/bin:$PATH
```

Save this change and exit the editor.

The contents of this file are executed by the shell each time you log into your computer. Since you have just created the file, your PATH has not yet been reset. Log out and then log back in again, and your PATH should be set. To check it, type

```
echo $PATH
```

This will print your PATH.

2.6 Compile and install legofit

The details are in the Legofit documentation.

3 Setting up your environment at the CHPC

These instructions are needed only if you’re using the facilities of the Center for High Performance Computing (CHPC) at the University of Utah. We are in the process of getting that set up for the workshop. The current instructions may need to change once we get that set up.

3.1 Soft links to group storage

In the sections that follow, I will introduce you to several directories and files on the CHPC servers. Each one is described by its "path name" relative to your own "home directory," which is represented by the symbol `~`. Before these path names will work, you'll need to define two "soft links" within your own home directory, which point to group storage devices owned by the Rogers lab. After you define these soft links, `~/grp1` will refer to one of these group storage devices and `~/grp2` to the other. You can establish these links by typing

```
ln -s /uufs/chpc.utah.edu/common/home/rogersa-group1 ~/grp1
ln -s /uufs/chpc.utah.edu/common/home/rogersa-group2 ~/grp2
```

If everyone defines these soft links in the same way, then we can all use path names of the form `~/grp1/rogers/data/whatever`.

4 Making an input file in .raf format

Legofit uses data in ".raf" format. The following data sets are already available on the server at Utah's CHPC.

Altai Neanderthal `~/grp1/rogers/data/altai/orig2/altai.raf.gz`

Vindija Neanderthal `~/grp1/rogers/data/vindija/orig/vindija.raf.gz`

Denisovan `~/grp1/rogers/data/denisova/orig2/denisova.raf.gz`

Western Europeans (SGDP) `~/grp1/rogers/data/simons/raf/weur.raf.gz`

You can use any or all of these in your projects. But you'll also need to make at least one more. To do so, copy the following script into a directory of your own:

```
~/grp1/rogers/data/simons/raf/weur.slr
```

Change its name (because "weur" stands for "western European") and edit it to reflect the population or populations that you want to study. It's set up to run on the "notchpeak" cluster, where I have only one node.

If that node is free, you can run it there by using "cd" to move to the directory that contains the script and then typing

```
sbatch <your script name>
```

where `<your script name>` is the name of your version of the script. This script will take many hours to complete. You can check on its status at any time by typing

```
squeue -u <your user id>
```

where `<your user id>` is the id you use to log onto the CHPC cluster. As a shorthand, I often type this as

```
squeue -u ‘whoami‘
```

which uses Linux's "whoami" command to fill in your user id.

If the notchpeak node is busy, try kingspeak instead. I have six nodes there, so your odds are better. Before doing so, edit your slurm script. The lines that read

```
#SBATCH --account=rogersa-np
#SBATCH --partition=rogersa-np
```

work on notchpeak but not on kingspeak. For kingspeak, they should read

```
#SBATCH --account=rogersa-kp
#SBATCH --partition=rogersa-kp
```

5 Tabulating site patterns

Legofit works with the frequencies of “nucleotide site patterns,” which are explained in this section of the Legofit website. During the workshop, you will not need to tabulate site patterns, because I’ll do that for you before the workshop begins. This section is here to help you do the job on your own, after the workshop.

The Legofit package includes several programs for tabulating site patterns. I’ll focus here on “sitepat,” which works with data files rather than with the output of a simulation program. Sitepat reads a series of files in .raf format and writes a file describing the frequency of each site pattern. Optionally, it also writes a series of output files, each describing site pattern frequencies in a different bootstrap replicate. These replicates are used for measuring statistical uncertainty. Details are in the Legofit docs. Sitepat reads enormous data files and can take hours to run, even on a compute cluster. That is why we aren’t doing this step during the workshop.

Have a look at file data.opf. After the header, the first few lines look like

```
#       SitePat          E[count]          loBnd          hiBnd
              x    312922.8110121 311974.2398439 313732.1909228
              y    306301.1413693 305178.5610866 307121.6833708
              v    102039.5699405 101293.0904018 102887.7097098
              a     78653.9538690  77779.2808408  79467.4795387
              d    339489.6889881 337728.6566221 341531.9807664
            x:y    189532.2514885 187871.4715776 191321.2242190
```

The left column lists the site patterns. “E[count]” is (more or less) the number of sites exhibiting each site pattern. (For a full explanation, see the Legofit docs.) The values under “loBnd” and “hiBnd” enclose a 95% confidence interval. The frequency of site pattern “x” is its value of “E[count]” divided by the sum of all the values under “E[count].”

It’s best to run sitepat within a script, both because it takes awhile to run, and also because the script will document what you did. Here is the slurm script that generated the output above.

6 Fitting models of history

For each project, I create a directory tree with the same format, which is illustrated in the github repository at legofit/europe. Within this directory, you’ll find (1) a README.md file, which describes all the others, (2) a file called “data.opf,” which contains observed site pattern frequencies, (3) sitepat.slr, a slurm script that runs “sitepat” and generates “data.opf,” and “boot,” a directory containing bootstrap replicates. There will eventually be other files with names like “all.bootci” and “all.bma,” which contain the results of analyses. Within this directory, you will also find subdirectories with names like “a,” “ab,” and so on. Each of these refers to a different model of history, which we wish to fit to the data in “data.opf.” Name these as you wish, but it’s a good idea to keep the names short. In my naming scheme, directory “a” contains a model in which there is only one episode of admixture, labeled α; "ab" is for a model that has episodes α and β; and so on.

Within the subdirectory for each model, assumptions about population history are described in file "a.lgo." These assumptions are described using syntax that you can read about here. Use `cd` to move into directory "a," and have a look at a.lgo. Then type this at the bash command line:

```
legosim a.lgo
```

This runs the program "legosim," which reads file a.lgo, estimates the probability of each site pattern, and prints out the result. By default, legosim doesn't calculate probabilities for singleton site patterns (those in which the derived allele is present only once), and it uses a stochastic algorithm to estimate probabilities. To include singleton site patterns and use the (faster and more accurate) deterministic algorithm, the command would be

```
legosim -1 -d 0 a.lgo
```

Because it is fast, the legosim program is a useful way to check for errors in your .lgo file. For further details about the program and its command-line arguments, see its documentation.

Having ascertained that the sotware can read "a.lgo" without errors, let's use the "legofit" program to estimate parameters. The parameters to be estimated are those defined as "free" in file "a.lgo." Type the following at the bash command line:

```
legofit -1 -d 0 --tol 1e-3 a.lgo ../data.opf
```

After a few seconds, legofit should print a page of output. But before examining that, let's unpack the command that generated it. The 1st argument on the command line is "`-1`." This tells legofit to use singleton site patterns—those in which the derived allele is present only once. Next, the arguments "`-d 0`" tells legofit to use its deterministic algorithm, which is faster and more accurate than the stochastic one it uses by default. (The stochastic algorithm is needed for complex models.) The arguments "`--tol 1e-3`" tell legofit to be satisfied with a fairly loose fit between model and data. In research, you would want a smaller number here. The next argument, `a.lgo`, is the file describing the model of history, and the final argument, `../data.opf`, is the data file.

Now lets examine the output of that last command. The first section of output echoes the the legofit command and the input parameters, along with default values of parameters that we didn't set on the command line.

```
########################################
# legofit: estimate population history #
#      version 2.3.8-12-gf1f00c9       #
########################################
#
# Program was compiled: Jun 21 2022 11:28:40
# Program was run: Thu Jul  7 20:38:55 2022
#
# cmd: legofit -1 -d 0 --tol 1e-3 a.lgo ../data.opf
# curr dir: /Users/rogers/txt/agar22/legofit/europe/a
# Stage nOptItr nSimReps
#     0    1000        1
# algorithm          : deterministic
# ignoring probs <=  : 0
# Branch length floor: 0
# DE strategy        : 2
#    F               : 0.3
#    CR              : 0.8
#    tolerance       : 0.001
# nthreads           : 2
# lgo input file     : a.lgo
# site pat input file: ../data.opf
# free parameters    : 12
# points in DE swarm : 120
# Including singleton site patterns.
# cost function      : KL
```

In the output above, "tolerance" is the value we specified as `--tol 1e-3` on the command line. The next section prints the initial values of all parameters, as read from the .lgo file.

```
Initial parameter values
Fixed:
      zero = 0
       one = 1
       TmN = 1
     Txynd = 25920
Free:
       Tnd = 24000
       Tav = 14000
       Txy = 10000
        Td = 2500
        Ta = 4000
        Tv = 1000
    twoNav = 2000
     twoNn = 2000
    twoNnd = 2000
    twoNxy = 20000
  twoNxynd = 20000
        mN = 0.01
```

Then comes a line that tells us the result of the optimization procedure.

```
DiffEv reached_goal. cost=7.22802e-04 spread=8.77863e-04
```

To understand this line, you need to know how legofit fits parameters to data. It uses something called "KL divergence" [2] to measure the difference between observed site pattern frequencies and those predicted by the model of history. Legofit searches for parameter values that minimize KL divergence, which is called "cost" in the output above. The value printed there (7.22802e-04) is the lowest KL divergence legofit was able to find. This search is conducted by an algorithm called "differential evolution" (DE) [3]. DE maintains a swarm of points, each representing a guess about parameter values. In each iteration of the algorithm, the best points (those with lowest KL) mutate, recombine, and reproduce to produce a new generation of points. The algorithm stops when the difference between the best point and the worst (called "spread" in the output above) falls to a specified tolerance. As you can see in this output, spread is smaller than 0.001, the tolerance we specified using "`--tol 1e-3`" on the command line. Thus, the algorithm has reached its goal, and this is why it printed `DiffEv reached_goal` in the output above. This goal is arbitrary, and the output can be useful even when the goal is not reached.

The next section of output lists the fitted values of all parameters.

```
Fitted parameter values
Free:
       Tnd = 24871.7
       Tav = 16727.1
       Txy = 13090.4
        Td = 1494.04
        Ta = 5346.73
        Tv = 2660.19
    twoNav = 14309.4
     twoNn = 8788.21
    twoNnd = 2998.19
    twoNxy = 23935.7
  twoNxynd = 46699.2
        mN = 0.0198353
```

Finally, legofit prints the expected branch length associated with each site pattern in the fitted model.

```
#       SitePat  BranchLen
              x 38204.0073508
              y 37200.1770458
              v 12628.3146863
              a 10003.0899715
              d 41976.1547361
            x:y 22591.0958761
            x:v 274.3861485
            x:a 278.8300856
            x:d 3121.4558596
            y:v 383.1800171
            y:a 321.8630482
            y:d 2936.3292837
            v:a 28571.5881777
            v:d 1060.8437350
            a:d 1067.6185093
          x:y:v 576.2188433
          x:y:a 571.7749063
          x:y:d 7001.3957667
          x:v:a 2884.5519560
          x:v:d 225.8439349
          x:a:d 230.2878720
          y:v:a 3460.0030731
          y:v:d 235.9020933
          y:a:d 229.1273190
          v:a:d 17599.3335385
```

```
    x:y:v:a 7238.2996703
    x:y:v:d 624.7610570
    x:y:a:d 620.3171199
    x:v:a:d 5567.0327885
    y:v:a:d 6019.8141157
```

For example, the expected branch length of site pattern x is 38204.0073508 generations. This is the average length, within the gene genealogy, of the branch that, should a mutation strike it, would generate the x site pattern. Under the model of infinite sites [1], the probability of a site pattern is proportional to its expected length and can be calculated as the ratio of this length to the sum of all the lengths.

6.1 Studying a model of history

For any given data set, you will want to study several models. It is convenient to keep the files relating to a given model in a directory just under the one that holds the data file. The directory for each model contains the following files:

`a.lgo` describes the model of history under study. The syntax of a .lgo file is described in the legofit documentation.

`a1.slr` is a script, which can be interpreted by "slurm," a program that manages jobs on a compute cluster. This file runs legofit on the real data.

`a1boot.slr` a slurm script that runs legofit on a bootstrap replicate. This script is run once for each bootstrap replicate, using slurm's `sbatch --array` command.

`a2.slr`

```
sed -i "" 's/5000/100/g' *.sh
```

7 Model selection and model averaging

8 Graphing results

References

[1] Motoo Kimura. "The Number of Heterozygous Nucleotide Sites Maintained in a Finite Population Due to Steady Flux of Mutation". In: *Genetics* 61 (1969), pp. 893–903. DOI: 10.1093/genetics/61.4.893.

[2] Solomon Kullback and Richard A Leibler. "On Information and Sufficiency". In: *The Annals of Mathematical Statistics* 22.1 (Mar. 1951), pp. 79–86.

[3] Kenneth Price, Rainer M Storn, and Jouni A Lampinen. *Differential Evolution: A Practical Approach to Global Optimization.* Berlin: Springer Science and Business Media, 2006. ISBN: 978-3-540-20950-8.

[4] Alan R. Rogers. "Legofit: Estimating Population History from Genetic Data". In: *BMC Bioinformatics* 20 (2019), p. 526. DOI: 10.1186/s12859-019-3154-1.

[5] Alan R. Rogers. "An Efficient Algorithm for Estimating Population History from Genetic Data". In: *Peer Community Journal* 2 (2022), e32. DOI: 10.24072/pcjournal.132.