A Legofit Tutorial

Alan R. Rogers

June 21, 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 [1, 2]. The full documentation is available online.

2 Installing and Using Legofit

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`. In that case, install git:

```
brew install git
```

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 git@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.

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

2.8   Making a 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
```

2.9 Working with Legofit

2.9.1 Organizing the top-level directory for a Legofit project

For each project, I create a directory tree with the same format.

The top-level directory contains the following files:

README.md Describes how all the other files in the directory were made.

data.opf Observed site-pattern frequencies, as generated by Legofit's "sitepat" program, under the control of "sitepat.slr."

boot A subdirectory containing bootstrap replicates, which have names like "boot0.opf," "boot1.opf," and so on. These files are also generated by "sitepat," under the control of "sitepat.slr."

sitepat.slr A slurm script that runs "sitepat" and creates "data.opf" and all the .opf files within subdirectory "boot."

Eventually, the top-level directory will contain other files, with names like "all.bootci" and "all.bma," which contain the results of analyses.

Within the top-level directory, there should be a directory for each model that is fit to the data in "data.opf" and "boot/boot*.opf." Name these as you wish, but it's a good idea to keep the names short. Mine tend to have names like "a," "b," "ab," etc. The "a" directory contains a model in which there is only one episode of admixture, labeled α. The "ab" directory is for a model that has episodes α and β.

2.9.2 Organizing the subdirectory for each model

To be continued.

References

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

[2]   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.