Lagrange C++ Manual

Stephen A. Smith and Richard Ree

Part I. Introduction

1 Impetus

Lagrange began as a Python library or program (however you want to look at it) with Richard Ree. I was coding along side this project a Java implementation (called AReA), but we later joined forces and produced the paper with the explicitly solved rate matrix, Q (Ree and Smith, 2008). This package has been successfully used for a variety of projects including some of our own. However, development has continued. Rick developed a nice web interface for the construction of input files. This is especially helpful when describing more complex area connectivity scenarios. Nevertheless, some users, including the developers, have recognized the speed limitations, and therefore dataset limitations, of the Python version. In order to continue the development of more computationally intensive procedures, we have moved to a C++ version. There are/will be ways to use the Python files created from the website, but you may also interact directly with the C++ program.

2 What is here

In this document you will find descriptions of the 1) compilation and installation process, 2) configuration file construction, 3) some examples and 4) some of the algorithmic advances.

Part II. Compilation and Installation

The C++ version of Lagrange makes use of quite a few libraries to simplify the code and limit the introduction of errors. The hardest part of compilation will be getting those installed and the difficulty will vary depending on the machine. In some cases (Windows and certain versions of Mac and Linux), we do not have access to machines in which to test compilation so you will be on your own. But please email with problems if they arise and we will try our best to resolve those.

A more extensive discussion of the Makefile options is also below.

The libraries used in Lagrange include:

- Gnu Scientific Library (GSL) – optimization
- armadillo – linear algebra (which requires cmake)
- gmp – optional, for calulating exceptionally large trees
- mpfr – optional, for calculating exceptionally large trees
- gmpfrxx – optional, for calculating exceptionally large trees

You will also have to be able to compile with a FORTRAN compiler (I have been using gfortran).

3 Linux

Compilation of Lagrange on a Linux machine will be the least painful. This is mostly because both Rick and I develop on Linux machines and so this is the operating system on which the most testing has occurred. In particular it has been tested on both Ubuntu and Fedora distributions. Because of the package management system on these machines, the vast majority of the relevant packages can be installed from the package managers. After downloading the packages, the current procedure for installation on Ubuntu 10.04 is:

1. make sure that you have the basic compilation tools (`sudo apt-get install build-essential`)
2. untar and install armadillo (contained in the downloaded package, this is newer than the repository version)
   (a) this may require the installation of other packages (for example, cmake, which can be installed with the command `sudo apt-get install cmake`)
   (b) to get all the packages necessary you can run `sudo apt-get build-dep armadillo`
3. install the gsl development libraries with the command `sudo apt-get install libgsl0-dev`
4. install the python development libraries so you can read in the web configuration files with `sudo apt-get install python2.6-dev`
5. if you are going to use the big tree option for compilation (good for trees over 1000 tips) you should do the following:
   (a) in the gmpfrxx directory, untar the gmp-4.3.1.tar.bz2 file and go inside and install with `sudo apt-get install m4`, `./configure --enable-cxx`, `make`, `sudo make install`)
   (b) untar the mpfr-2.4.2.tar.gz and do the standard `./configure`, `make`, `sudo make install`)
   (c) finally go back into the gmpfrxx and just type `make`
6. make lagrange with the command `make -f Makefile`

4 MacOSX

Most of the installation instructions are similar to Linux for MacOSX however there are some differences.

1. install the developer tools for MacOSX (these should be on the disk that comes with the computer or you can always download from apple. You will need this to install anything from source code on the Mac. You will also need cmake for armadillo (see below).
2. untar and install armadillo (contained in the downloaded package)
   (a) there are special instructions for the Mac so please look at the armadillo README.txt within the directory
3. install the GNU Gnu Scientific Library which can be found here `http://www.gnu.org/software/gsl/`.
4. if you are going to use the big tree option for compilation (good for trees over 1000 tips) you should do the following:
   (a) in the gmpfrxx directory, untar the gmp-4.3.1.tar.bz2 file and go inside and install with `sudo apt-get install m4`, `./configure --enable-cxx`, `make`, `sudo make install`)
   (b) untar the mpfr-2.4.2.tar.gz and do the standard `./configure`, `make`, `sudo make install`)
   (c) finally go back into the gmpfrxx and just type `make`

5. make lagrange with the command `make -f Makefile.MAC`

When available, a binary version of Lagrange is made available for Mac users to bypass the need to compile it from source.

5 Windows

Currently being test on a Windows 7 machines, but check out the instructions for Linux and Mac and use CygWin.

6 Makefile description

There are some configurable items in the Makefile for each system that you should be aware of in case somethings goes wrong. Below are some lines of which to be aware.

6.1 Linux

```
#if you have an error related to python during compilation, check this line
PYTHON_LIB = -I/usr/include/python2.6/
# requires fortran, gsl, and pthread – and -ldl -lutil -lpython2.6 are for python
# if llapack lblas fail, try larmadillo
LIBS := -llapack -lblas -lgfortran -lgsl -lgslcblas -lm -lpthread -lgmp -ldl -lutil -lpython2.6
##########
# change to yes for bigtrees – loses about 3x speed
# if 64 bit GSL try CPPFLAGS="-arch x86_64" LDFLAGS="-arch x86_64" ./configure
# need to install gmp (with ./configure –enable-cxx) and mpfr and gmpfrxx
#######
BIG = no
#this is where the fortran compiler should be
FC = /usr/bin/gfortran
```

6.2 MacOSX

```
#if you have an error related to python during compilation, check these lines
PYTHON_LIB = -I/System/Library/Frameworks/Python.framework/Headers/
#output of
#>>> import distutils.sysconfig
#>>> distutils.sysconfig.get_config_var('LINKFORSHARED')
PYTHON_REQ = -u _PyMac_Error /System/Library/Frameworks/Python.framework/Versions/2.6/Python
##########
# change to yes for bigtrees – loses about 3x speed
# if 64 bit GSL try CPPFLAGS="-arch x86_64" LDFLAGS="-arch x86_64" ./configure
# need to install gmp (with ./configure –enable-cxx) and mpfr and gmpfrxx
#######
BIG = no
#this is where the fortran compiler should be
FC = /usr/bin/gfortran
```

Part III. Configuration file

In order for Lagrange to run it must be informed of your tree and data and parameters. This is done in a plain text file the options for which are described below.

7 Typical configuration file

The configuration file is meant to have an extremely simple format where there is a `parameter = option` format.

So an example configuration file might look like:

```
treefile = aesculus.tre
datafile = aesculus.data
areanames = EA ENA WNA EU
ancstate = _all_
states
```

In this example, Lagrange will look for a treefile named aesculus.tre in the current directory and a datafile named aesculus.data in the current directory. It will them assume that the area names corresponding to the data in the aesculus.data file are called EA ENA WNA and EU respectfully. It will then assume that you want ancestral range caluclations for all the nodes and it will report the states at the nodes (as opposed to splits which will report the splits at the nodes). This is a simple file that might serve many of the needs of people using Lagrange.

The treefile is assumed to contain newick tree(s) alone. If more than one tree is in the file then it will assume that you want to reconstruct across all those trees and will do so. There is no other option for conducting the analyses on multiple trees. In the case of Bayesian reconstructions, it will sample from those trees for reconstruction ancestral ranges or for stochastic mapping. An example treefile might look like:

```
((a:0.1,b:0.1):0.2,c:0.3);
```

Or in the case of many trees:

```
((a:0.1,b:0.1):0.2,c:0.3);
((a:0.1,b:0.1):0.15,c:0.25);
((a:0.1,c:0.1):0.15,b:0.25);
```

As a note, there should be no translation tables and of course the names have to correspond to the datafile.

The datafile is also meant to be simple. It is simply an extended Phylip format. An example would be:

```
3 4
a 1100
b 0110
c 0011
```

The areanames are defined in the configuration file so there is no need to place them here. In the example above we might define in the configuration file that `areanames = EA ENA WNA EU`.

With the two important bits being described (treefile and datafile), we can now move on to other configuration options in the configuration file. The two that need descriptions are the area connectivity options

and period disgnation options. In the configuration file, these two may look like:

```
ratematrix = aesculus.rm
periods = 10 100
```

The first option is pointing to a file called aesculus.rm which contains the rate matrix connectivity parameterization. If no file is designated, then all connectivities will be considered or unconstrained. If a connectivity scenario is desired, then the file might look like:

```
1 1 1 1
1 1 0.1 1
1 0.1 1 1
1 1 1 1


1 1 1 1
1 1 0.2 1
1 1 1 1
1 1 1 1
```

This says that for the first period (from 0 to 10) there is less dispersal between areas 2 and 3 and in the second period there is only less dispersal from 2 to 3.

The second option (`periods = 10 100`) designates what time slices or periods will be considered. This is important when designating the above matrix if different connectivities are used for different times, and it can also be important when estimating the rate matrix parameters if you want different estimates for different time periods.

Other options are fairly simple and are described below.

- `mrca = name species1 species2` – this is used when wanting ancestral state results for only particular nodes, these nodes need to be specified. This is not necessarily very useful now because the forward-reverse algorithm implemented now does not require much extra time to report results for all nodes. I might just leave it alone and work on parse the results for the nodes of interest instead of defining here.

- `fixnode = mrcaname distribution` – example: `fixnode = ROOT 1000` – this is used, with a mrca definition as above in order to fix the ancestral distribution of a node.

- `excludedists = distributions` – example: `excludedists = 1111` – this is used to exclude a set of distributions from the model. More than one model may be seperated by commas.

- `includedists = distributions` – example: `includedists = 1000,0100,0010,0001,1100,1010,1001,0110,010` – this is used as the opposite of above, where you only want to include a certain list of distributions from the model.

- `includedists = number` – example: `includedists = 2` – when the above option is followed by a number, it will only include distributions that include that number of areas or less.

- `ancstates = nodes` – example: `ancstates = ROOT` – this designates that you want ancestral states to be calculated for a node. To do this for all nodes, simply use `ancstates = _all_`.

- `states|splits` – example: `states` – this designates what you want reported for ancestral calculations, either states or splits. Splits will give the particular speciation split with the highest likelihood, while states will give the most likely range or state.

- `estimate_dispersal_mask` – this designates that you want the entire dispersal mask to be estimated (every dispersal rate).

Part IV. Python Lagrange configuration

In order to utilize this, you must go to the website here.