Linking to Native Routines in This Package

Charles J. Geyer

December 7, 2025

1 Aster Models

Aster models implemented in this package (`aster`, Geyer, 2017a) are described by Geyer, Wagenius and Shaw (2007a) but are better described by the first draft of that paper (Geyer, Wagenius and Shaw, 2007b, Chapter 1). or by the course slides for a course on aster models `http://www.stat.umn.edu/geyer/8931aster/slides/s2.pdf`.

The issue is that Geyer et al. (2007a) describe too many aster models, those implemented in the package `aster2` (Geyer, 2015), and while discussing this package we don't want to know about aster models it does not implement.

The aster models in this package are described by

(a) A directed acyclic graph in which each node has at most one predecessor.

(b) A one-parameter exponential family of distributions associated with each arrow of this graph.

(c) A data vector giving data for each non-initial node of the graph for each individual.

(d) A data vector giving data for each initial node of the graph for each individual.

A node is *initial* if it has no predecessor, otherwise *non-initial*. This agrees with the terminology used in the `aster2` package and recent papers and technical reports about aster models by this author. It disagrees with the terminology used in the `aster` package which says *root* instead of *initial*.

In this package, every individual has the same graph. Call the number of non-initial nodes in that graph `nnode`. Give these non-initial nodes indices

(integers from 1 to `nnode`) so that each predecessor has a lower index than any of its successors. Then the graph for one individual is specified by an integer vector of length `nnode`, which gives for each non-initial node the index of its predecessor node if its predecessor is non-initial and zero otherwise (meaning its predecessor is an initial node). Call this vector `pred`. It specifies (a) in the list above.

Item (b) in the list above is also specified by an integer vector of length `nnode`. Call it `fam`. The specification also needs a mapping from integers to families. The families for the model are specified by a list of R objects of class `"astfam"`, which are described by the help page

```
library("aster")
help("families")
```

Call this list `famlist`. Then these must satisfy

```
all(fam %in% seq(along = famlist))
```

Then `famlist[[fam[j]]]` specifies the family associated with the arrow to node `j` from its predecessor.

Data at non-initial nodes is considered random and is specified by a `double` vector of length `nind * nnode`, where `nind` is the number of individuals on which we have data. The order in this vector is first node of the graph for all individuals, second node of the graph for all individuals (in the same order as before), and so forth. Call this vector `resp`.

Data at initial nodes is considered non-random and is specified by a `double` vector of length `nind * nnode`. The order in this vector is the same as for `resp`. Call this vector `root`. The meaning of this vector is that if we turn `resp` and `root` into matrices

```
resp <- matrix(resp, nind, nnode)
root <- matrix(root, nind, nnode)
```

so `resp[i, j]` is the data for individual `i` and node `j`, then the arrow in the graph going to node `j` (there is exactly one by assumption) has successor data `resp[i, j]` for individual `i` and predecessor data

(e) `resp[i, pred[j]]` in case `pred[j]` is not zero, and

(f) `root[i, j]` in case `pred[j]` is zero.

That specifies the aster model and its data.

Now we have parameters. The *conditional canonical parameterization* of the saturated aster model is a vector `theta` laid out like `resp` and `root`.

If we also consider it a matrix (like we did for `resp` and `root` above), then `theta[i, j]` is the parameter for the conditional distribution of `resp[i, j]` given its predecessor data (specified by (e) or (f) in the list above as the case may be).

We also have a parameter vector `phi` which is laid out like `theta`, `root`, and `resp` called the *unconditional canonical parameter vector* of the saturated aster model. This is too complicated to describe here. See Section 2.3 of Geyer et al. (2007a) or (better) Section 1.1.3 of Geyer et al. (2007b) or (better still) the aforementioned course slides (slides 1–37 of `http://www.stat.umn.edu/geyer/8931aster/slides/s2.pdf`).

Everything in this section agrees with this package and its documentation. The other descriptions are just better descriptions of the same thing.

Summary A saturated aster model is specified by vectors `pred`, `fam`, and `famlist` described above. Its data is specified by vectors `resp` and `root` described above. One particular distribution in the model is specified by a parameter vector, one or the other of `theta` and `phi` described above.

2 Evaluating the Aster Log Likelihood in C in Another Package

This package registers two C functions via the `R_RegisterCCallable` mechanism described in Section 5.4.2 of *Writing R Extensions* (R Development Core Team, 2017). Their prototypes (found in `mlogl.h` in the `src` directory) are

```
double aster_mlogl_sat_unco(int nind, int nnode, int *pred, int *fam,
    double *phi, double *root, double *response, _Bool check);

double aster_mlogl_sat_cond(int nind, int nnode, int *pred, int *fam,
    double *theta, double *root, double *response, _Bool check);
```

and a correct typedef for these functions is found in the `mlogl-export.h` file in the `inst/include` directory, which, of course, is in the `include` directory when this package is installed

```
typedef double (*aster_mlogl_sat_either_funptr)(int nind, int nnode,
    int *pred, int *fam, double *phi, double *root, double *response,
    _Bool check);
```

To call one of these functions (for specificity, say the former) from C code in another package, one does the following. For a toy working example of this see the demonstration packages in the `linkingTo` git repository Geyer (2017b). This discussion is copied from that.

1. The calling package (the one you write) must have `aster (>= 0.9)` in either the `Depends` or the `Imports` field of its `DESCRIPTION` file.

2. The calling package (the one you write) must have `\import{aster}` in its `NAMESPACE` file. Or perhaps only import some functions from aster in the `NAMESPACE` file with an `\importFrom` directive, as described in Section 1.5.1 of *Writing R Extensions* (R Development Core Team, 2017). The purpose of this item and the preceding one is to have R package `aster` loaded before your package (so its code is available to yours).

3. The calling package (the one you write) must have `aster (>= 0.9)` in the `LinkingTo` field of its `DESCRIPTION` file. The purpose of this item is to have the include file `mlogl-export.h` in R package `aster` available to your package, that is,

   ```
   #include "mlogl-export.h"
   ```

   will work in C code in your package.

Now in your package, you can write a function, call it `mlogl`, that has prototype the same as the function you are calling, that is,

```
double mlogl(int nind, int nnode, int *pred, int *fam,
    double *theta, double *root, double *response, _Bool check);
```

Say you put that in a header file called `mlogl.h`.

Then the following code should work

```
#include <stddef.h>             // defines NULL
#include <R_ext/Rdynload.h>     // defines R_GetCCallable
#include "mlogl-export.h"       // defines aster_mlogl_sat_either_funptr
#include "mlogl.h"              // defines mlogl

inline double mlogl(int nind, int nnode, int *pred, int *fam,
    double *phi, double *root, double *response, _Bool check)
```

```
{
    static aster_mlogl_sat_either_funptr fun = NULL;
    if (fun == NULL)
        fun = (aster_mlogl_sat_either_funptr)
            R_GetCCallable("aster", "aster_mlogl_sat_unco");
    return fun(nind, nnode, pred, fam, phi, root, response, check);
}
```

Now there is one last thing to say. What happened to `famlist`?

4. Before any calls to `mlogl` one must call from R (before going to C)

   ```
   aster:::setfam(famlist)
   ```

   This sets the identification of integers with families.

5. Then one can call `mlogl` (from C) as many times as one likes.

6. Finally (after returning to R from C) one should call (from R)

   ```
   aster:::clearfam()
   ```

   This clears the identification of integers with families.

References

Geyer, C. J. (2015). R package `aster2` (Aster Models), version 0.2-1. `http://www.stat.umn.edu/geyer/aster/` and `https://cran.r-project.org/package=aster2`.

Geyer, C. J. (2017a). R package `aster` (Aster Modeels), version 0.9. `http://www.stat.umn.edu/geyer/aster/` and `https://cran.r-project.org/package=aster`.

Geyer, C. J. (2017b). Github repo `linkingTo`, which contains two R packages `foompter` (version 0.2) and `goompter` (version 0.2) illustrating calling C functions from one from C functions called from R in the other. `https://github.com/cjgeyer/linkingTo`.

Geyer, C. J., Wagenius, S. and Shaw, R. G. (2007a). Aster models for life history analysis. *Biometrika*, **94**, 415–426.

Geyer, C. J., Wagenius, S. and Shaw, R. G. (2007b). Aster models for life history analysis. Technical Report No. 644, School of Statistics, University of Minnesota. http://www.stat.umn.edu/geyer/aster/tr644.pdf.

R Development Core Team (2017). *Writing R Extensions*. https://cran.r-project.org/doc/manuals/r-release/R-exts.html. Also available in PDF or EPUB format.