Design Document for the Aster Package

Charles J. Geyer

July 13, 2006

1 Current Status

Versions of the aster package up through version 0-4.1, which at the time this document was begun was version on CRAN, were written without a design document, and it shows, the package having a number of limitations and other problems. At that time, the development version of the aster package differed from the CRAN version only by the addition of another family `two.truncated.poisson`. None of the design issues had been addressed.

Known design issues (bugs, misfeatures, whatever) are as follows.

1. Representation of families.
2. Representation of graphical model.
3. Data and parameter validity checks.
4. Distinguished point in parameter space.
5. Chain components (multivariate families).
6. Elimination of "individuals".
7. Starting point for optimization.
8. Constrained parameter spaces.
9. Mixed parameterization.
10. User-specified families.
11. Identifiability.
12. Maximization with respect to non-exponential family parameters.

We take these out of order.

2 Elimination of Individuals

This is item 6 on our list. Theoretically, it is trivial. The first submission of Geyer, Wagenius, and Shaw (2005) had a notion of "individuals." Data were indexed X_{ij} where i ran over individuals and j ran over nodes of the graph. In the second submission it was realized that individuals were superfluous, the index i could be removed from the notation, and data indexed X_j. The graph indicates the dependence structure, including the independence of "individuals." So individuals were a mere convenience. Moreover, that convenience made the notation (and the computer code) messier and confused the referee. Models with individuals appeared less general, although they were actually not: the case of an old-style model with precisely one individual is the same as a new-style model, which has no notion of individual.

Hence we could make no code changes at all. Individuals are sometimes a convenience and do no harm, because they can be avoided. But eventually the code should be changed to match the final version of the paper (whatever that may look like).

In particular, we should simply eliminate all references to individuals at all user-visible levels. In the R functions, in their documentation, in the package vignette. In any new vignettes we write. That is a lot of work.

We could also eliminate all reference to individuals in the C code, but that is perhaps not worth the trouble. A bunch of

```
for (i = 0; i < nind; ++i)
```

loops that always execute exactly once because we always have `nind` equal to one, don't slow anything down. They are just annoying to read. Since few people will read the C code, it doesn't really matter.

Eventually, the C code will also have to be rewritten to account for chain components (Section 5 below). There is no point in two rewrites (one to eliminate individuals, one to introduce chain components). We should do both at once.

3 Affine Models

This is item 4 on our list, but it also relates to items 7 and 8. The conditional canonical parameter space Θ or the unconditional canonical parameter space Φ are convex sets in $\mathbb{R}^d$ (Appendix E of Technical Report 644). All versions of the package to date have assumed $\Theta = \Phi = \mathbb{R}^d$, since that was the

case for the families implemented to date (Bernoulli, Poisson, k-truncated Poisson). Now we want to add negative binomial and k-truncated negative binomial, for which the (one-dimensional) parameter space is $\{\, \theta : \theta < 0 \,\}$. Hence we run up against constraints, item 8 on our list, and the parameter space not containing zero, which is item 7 on our list, because the current versions have the zero vector as the default starting point for optimization (the user can specify another point, but the default should work). We need to fix those. For more constrained models, see Section 6 below.

More theoretical and ultimately more important is the notion of *canonical linear models*, which are described in Section 1.4 of the revised version of Geyer, Wagenius, and Shaw (2005), are reparameterizations of the form

$$\boldsymbol{\eta} = \mathbf{M}\boldsymbol{\beta} \tag{1}$$

where the "linear predictor" $\boldsymbol{\eta}$ is either the conditional canonical parameter vector $\boldsymbol{\theta}$ or the unconditional canonical parameter vector $\boldsymbol{\varphi}$ depending on which kind of model is being used, where $\mathbf{M}$ is a known matrix called the *model matrix*, and $\boldsymbol{\beta}$ is the new parameter vector. This is like conventional linear and generalized linear model theory, but we now see that it is completely bogus (and always was). The idea that if you want an "intercept" term you add a column of ones to $\mathbf{M}$ and thereby add another parameter (increase the dimension of $\boldsymbol{\beta}$) and that is the *only* way to get an "intercept" in the model is wrong-headed. For one thing, it assumes that the zero vector is always a valid parameter value. Now we see it isn't.

What we need instead is the notion of *canonical affine models*, which replace (1) by

$$\boldsymbol{\eta} = \mathbf{a} + \mathbf{M}\boldsymbol{\beta} \tag{2}$$

where $\mathbf{a}$ is a known vector, which we will call the *distinguished point* of the parameter space. Note that if $\mathbf{a}$ is in the column space of $\mathbf{M}$, that is, $\mathbf{a} = \mathbf{M}\boldsymbol{\beta}$ for some $\boldsymbol{\beta}$, then (1) and (2) specify the same family of models, and the extra generality of (2) does nothing. This is the usual case, and that explains the popularity of linear models.

Sometimes, however, we do not want our distinguished point $\mathbf{a}$ to be the zero vector. We will impose the condition that $\mathbf{a}$ is in the parameter space of the linear predictor, either Θ or Φ as the case may be. Then $\boldsymbol{\beta} = \mathbf{0}$ maps to $\boldsymbol{\eta} = \mathbf{a}$, and hence $\boldsymbol{\beta} = \mathbf{0}$ is again a valid starting point, which solves in generality item 7 on our list.

We need to see what replacing (1) by (2) does to the rest of aster model theory. Writing

$$\tilde{l}(\boldsymbol{\beta}) = l(\boldsymbol{\eta}) = l(\mathbf{a} + \mathbf{M}\boldsymbol{\beta})$$

we get

$$\nabla \tilde{l}(\boldsymbol{\beta}) = \nabla l(\boldsymbol{\eta}) \mathbf{M}$$
$$\nabla^2 \tilde{l}(\boldsymbol{\beta}) = \mathbf{M}^T \nabla^2 l(\boldsymbol{\eta}) \mathbf{M}$$

which are the same formulas as when we define the linear predictor by (1) except for $\boldsymbol{\eta}$ being defined differently.

Thus it seems the only changes we need to make to add canonical affine models are adding a "distinguished point" $\mathbf{a}$ argument to the `aster` function and changing the evaluation of the linear predictor $\boldsymbol{\eta}$.

It is an interesting historical note that my thesis makes a great deal of fuss about affine versus linear. I guess I forgot. That stuff wasn't published. It was too trivial. And it is trivial, but a triviality that conventional statistics botches. We can't afford the luxury if we are going to have general exponential families (not just ones whose canonical parameter space is the whole real line). I just checked the help for the R `family` function and note that `glm` doesn't handle negative binomial, hence avoids dealing with this issue.

The alternative of redefining the model (negative binomial, for example) so its canonical parameter space contains zero is unappealing. For one thing, the unconventional parameterization may confuse users. For another, although we can assure the conditional canonical parameter space Θ contains the zero vector, we cannot assure the unconditional canonical parameter space Φ contains the zero vector, because the mapping from Θ to Φ is very complicated.

When the user does not specify a distinguished point, there should be one provided by the software. This means adding a new function to family implementations, or perhaps just a new column in the family branch table, that provides a usable default, e. g., zero for the families for which that works, but perhaps -1 for the negative binomial. Collecting all such defaults into a vector gives a valid conditional canonical parameter value distinguished point $\boldsymbol{\theta}$. Mapping that to the corresponding unconditional canonical parameter value $\boldsymbol{\varphi}$ gives a valid distinguished point for unconditional models.

Note that this will change the behavior of unconditional aster models. It will change the analysis for the example in Geyer, Wagenius, and Shaw (2005), which was not changed in the revision! What do we think about that? Of course, we can always set the distinguished point so that the analysis does not change, that is, use $\mathbf{a} = \mathbf{0}$ rather than the image of $\mathbf{0}$ under the mapping $\boldsymbol{\theta} \mapsto \boldsymbol{\varphi}$.

What do we think about that? It does not seem to be a serious problem. If `varbl` was in the model, then we are in the case discussed above where $\mathbf{a}$ is in the column space of $\mathbf{M}$ and (1) and (2) parameterize the same families of distributions (though not with the same $\boldsymbol{\beta}$ in each). Although the "`varb`" regression coefficients change, the MLE *probability distribution* does not. So we are o. k. Nothing besides those few regression coefficients would change. The confidence intervals shown in Figure 2 would not change. The P-values for the tests shown in Table 1 would not change.

One last point about item 8 on our list. If we limit aster to full, regular exponential families, then if we start at a feasible point (which we will by default) and if the `mlogl` function returns `list(value = Inf)` when called with a parameter value not in the canonical parameter space (conditional or unconditional as the case may be), then the default optimizer (my new `trust` package) can handle this.

This only works for *regular* exponential families, whose full canonical parameter space is an open set, in which case minus the log likelihood converges to ∞ as the parameter goes to the boundary (a property of exponential families, the log likelihood is upper semicontinuous, so its minus, what `mlogl` computes, must either go to infinity or be finite on the boundary, by lower semicontinuity, and the latter is ruled out by the definition of regularity). So the solution is never on the boundary, and a trust region algorithm can deal with the boundary (by always decreasing the trust region whenever the value `Inf` is returned by `mlogl`).

Since `nlm` and `optim` cannot handle boundaries well (even this very well behaved kind). We might as well take them out and make `trust` the only allowed optimizer.

Fortunately, brand name exponential families are all regular. We need not worry about non-regular exponential families.

4 Representation of Families

This is items 1 and 10 on our list. It involves a very BAD (broken as designed, a hackerism) decision. In versions to date, a family is coded by an integer, which serves simultaneously as an index for the vector of names returned by `families()`, which is

```
[1] "bernoulli"         "poisson"           "non.zero.poisson"
[4] "two.trunc.poisson"
```

in the development version (and the first three in earlier versions) and also as an index into a branch table `myfuntab` found in `astfam.c`, which is a

static array of `struct funtab` allocated at compile time.

This is very inflexible. We can see it already in not being able to specify k-truncated Poisson for arbitrary k. (Why just zero and two? The code is written for general k, but there is no way to specify a family with an extra parameter k).

Now we want more families with extra non-exponential-family parameters. We will refer to them here as "hyperparameters" although our meaning (non-exponential-family) has nothing to do with the conventional Bayesian usage. Let us list some.

(a) k-truncated Poisson with k a nonnegative integer hyperparameter.

(b) negative binomial with size parameter α a positive real hyperparameter. If ψ is the cumulant function of the geometric family, then we mean the family with cumulant function $\alpha\psi$.

(c) k-truncated negative binomial with k a nonnegative integer hyperparameter and size α a positive real hyperparameter (the previous item truncated).

(d) untruncated Poisson with size parameter r a positive real hyperparameter. If ψ is the cumulant function of the usual Poisson, then we mean the family with cumulant function $r\psi$. This is equivalent to adding the constant $\log r$ to the conditional canonical parameter, what the `glm` function deals with by way of its `offset` argument.

(e) k-truncated Poisson with k a nonnegative integer hyperparameter and size r a positive real parameter (the previous item truncated).

(f) binomial with sample size n as a positive integer hyperparameter. If ψ is the cumulant function of the Bernoulli family, then we mean the family with cumulant function $n\psi$.

(g) normal location family with scale parameter σ a positive real hyperparameter.

(h) gamma scale family with shape parameter α a positive real hyperparameter. If ψ is the cumulant function of the exponential distribution, then we mean the family with cumulant function $\alpha\psi$.

This does not even get into multivariate (chain group) families, which are the subject of Section 5 below.

In this document we will call any of the above a *superfamily*. It becomes a *family* when all hyperparameters are specified. Negative binomial is a superfamily; negative binomial with size = 2.22 (and no truncation) is a family. We can consider Bernoulli both a family and a superfamily: since it has no hyperparameters, there is no difference.

One further issue, which is item 10. We should allow a new superfamily to be specified without recompilation. The user should be allowed to code the cumulant function and its first two derivatives in R as well as a random variate simulation function, and it should work (assuming the R code is o. k.) This is not a high priority, but we should allow for the possibility in our redesign. More on this in Section 10 below.

For our first redesign decision, do we keep integer codes for families or drop them entirely? Since in any particular analysis, we do not expect to use many different families, we can keep integer codes if we allow them to be associated with general family specifications, (like negative binomial with size = 2.22). But this means if we continue to use these integer codes as indices into a branch table, the branch table (1) cannot be statically allocated and (2) must have room for hyperparameter values, so (3) different rows of the table can correspond to families belonging to the same superfamily.

Actually (1) is overkill. It could be statically allocated with room for, say, 100 families, but no arbitrary limits is preferable. Or it could be dynamically allocated and never freed, although perhaps reallocated if more room is needed. Actually we should stop this blather and explain the issue we are struggling with here. There is no problem when the user calls `aster` to fit a model. Memory for the branch table could be allocated at the beginning and released at the end. The problem arises that the `mlogl` function need not be called from inside aster, and has some usefulness called from outside. (I have used it in the Lande-Arnold analysis to calculate Pearson residuals.) Actually it is not clear this is a problem.

As a user interface issue, we should make the `fam` argument to `aster` and `mlogl` the same. It is the R way, that these be informative, not integers. So they can just be passed to `mlogl`, which can allocate its own branch table. Arrrghhh!!!!

That was way too bogged down in implementation details. There is no problem in separating `mlogl` into two functions, one called from inside `aster` and one called from the user level. The user level `mlogl` should do its own branch table allocation and deallocation. The non-user level `mloglhelper`, which is not exported by the aster package and which is only called from inside the `objfun` defined inside `aster.default`, need not allocate-deallocate, since `aster.default` can handle that.

Anyway, allocation-deallocation versus static is not a priority. Go with static and error message for more than 100 families for now.

Similarly, we can go with a fixed upper bound on the number of hyperparameters for now. Looks like two would handle all we have in mind. Could allow more.

So this brings us to the important decision about how the user specifies families. For reasons of backward compatibility, we would like to allow the use of numbers with the default coding for numbers 1 to 4 being what we have now (in the development version, shown above). We should also allow family specifications something like

```
list(name = "negative.binomial", shape = 2.22, truncation = 2)
```

It is a user interface issue that if the `truncation` argument is missing, then this means the ordinary untruncated negative binomial. Similarly, we allow

```
list(name = "poisson")
list(name = "poisson", truncation = 0)
list(name = "poisson", shape = 2.22)
list(name = "poisson", shape = 2.22, truncation = 0)
```

By analogy, we could allow

```
list(name = "negative.binomial")
```

as an untruncated geometric (negative binomial with shape $= 1$) but perhaps we should also allow

```
list(name = "geometric")
list(name = "geometric", truncation = 2)
```

Probably we should provide constructor functions for these guys. We must choose names that do not conflict with the constructor functions for use with `glm` documented on the help for the `family` function.

```
fam.poisson(size = 1, truncation) {
    stopifnot(is.numeric(size))
    stopifnot(length(size) == 1)
    stopifnot(size > 0)
    if (missing(truncation)) {
        result <- list(name = "poisson", size = size)
    } else {
        stopifnot(is.numeric(truncation))
```

```
        stopifnot(length(truncation) == 1)
        stopifnot(truncation == round(truncation))
        stopifnot(truncation >= 0)
        result <- list(name = "poisson", size = size,
            truncation = truncation)
    }
    class(result) <- "astfam"
    return(result)
}
```

Providing constructor functions allows for defaults and arg matching so `fam.poisson(tr = 2)` works, as does `fam.poisson()`. It also allows for error checking, so the call `fam.poisson(size = - 6)` does not work. It also tags the result with class `"astfam"` so checks on that are possible further down the road. Of course users can fake all of this stuff, but if they do that, then *ipso facto* they are experts and deserve whatever happens to them.

At the top of `glm` we see

```
if (is.character(family))
    family <- get(family, mode = "function",
        envir = parent.frame())
if (is.function(family))
    family <- family()
if (is.null(family$family)) {
    print(family)
    stop("'family' not recognized")
}
```

Users can certainly store functions in a list, since they are objects like any other. We can certainly do the

```
if (is.function(family))
    family <- family()
```

trick. Presumably we replace the last bit with

```
if (class(family) != "astfam") {
    print(family)
    stop("'family' not recognized")
}
```

We can also do the trick with character strings. Of course those strings are not very natural, `fam.poisson` and the like. We could make `fam` bogo-generic and do something like

```
fam <- function(string) {
    stopifnot(is.character(string))
    famfunnam <- paste("fam", "string", sep = ".")
    famfun <- get(famfunnam, mode = "function")
    if (is.function(famfun)) {
        result <- try(famfun())
        if (class(result) == "astfam")
            return(result)
    }
    famfun <- get(string, mode = "function")
    if (is.function(famfun)) {
        result <- try(famfun())
        if (class(result) == "astfam")
            return(result)
    }
    print(string)
    stop("'family' not recognized")
}
```

Now both `fam("fam.poisson")` and `fam("poisson")` work. Note that it is important that we try with the `famfunnam` first, so we do not call the `poisson` function that goes with `glm`. If the argument string is `"poisson"`, then we succeed in looking up the function `"fam.poisson"` and return the result of that. In any event, since `poisson()` does not return an object of class `"astfam"`, we won't go completely wrong even if we reversed the order, so long as we check for the result being class `"astfam"`.

Now we can have at the top of `aster` or whatever needs it

```
if (is.character(family))
    family <- fam(family)
if (is.function(family))
    family <- try(family())
if (class(family) != "astfam") {
    print(family)
    stop("'family' not recognized")
}
```

We haven't resolved all issues about family representation, but to do that we need to also look at multivariate families. See Section 5.2 below.

5 Multivariate Families

This is item 5 on our list. Item 2 is also involved. There are two quite unrelated issues here, which we deal with in two subsections.

5.1 Representation of Graphs

To specify an aster-type chain graph model we need to have an index set J running over non-root nodes of the graph, which we may take to be `seq(1:nrow(data))`, where `data` is the data frame. Then we need a partition $\mathcal{G}$ of J, which we may represent as a list of lists, call it `G` in R. Then we must have

```
identical(sort(unlist(G)), seq(1, nrow(data)))
```

in order for `G` to represent a partition. Finally we need the parent map

$$p : \mathcal{G} \to J \cup F$$

where F is the set of root nodes. Let us keep the idea that index zero codes for root, as in current and earlier versions. Then we can represent p as an integer vector `pred` of length `length(G)` taking values in `seq(0, nrow(data))`. This says the parent of chain component `G[[i]]` is either node `pred[i]` or is a root node if `pred[i] == 0`, in which case we have to look up the data as `root[i]`. Note that this means the lengths of `pred` and `root` change from `nrow(data)` (which is what they are in all versions to date) to `length(G)`, that is, from the cardinality of J (the number of non-root nodes) to the cardinality of $\mathcal{G}$ (the number of chain components).

In versions up to the current version, when p was a map $J \to J \cup F$ so that every chain component (in the new terminology) was a singleton, we required

```
all(pred < seq(along = pred))
```

to ensure the graph was acyclic. This required the user to assure that parents came before (lower index) than children in the data. Now the situation is much more complicated. The analogous check for the new setup would be

```
foo <- TRUE
for (i in seq(along = G))
    foo <- foo && all(pred[i] < G[[i]])
```

After the loop `foo` should still be true (is there a way to do this more simply?) Of course, this is tricker to explain to users and even trickier for users to comprehend. Perhaps it is time to bite the bullet and implement a topological sort for R. A simple algorithm that tests a directed graph for acyclicity is given by Aho, Hopcroft and Ullman (1983, p. 221). The Unix function `tsort` tests a directed graph for acyclicity and produces the total ordering consistent with the partial ordering if the graph is acyclic. Curiously, although Unix has always had this main program `tsort`, neither Unix nor R has a library function to do this so-called "topological sort" algorithm. A search of `http://r-project.org` shows that the contributed package `ggm` has a topological sort function `topSort` but it only works on directed graphs represented as adjacency matrices (which we don't want to deal with). Topological sort is trivial. Just re-implement it. Eventually. It is not a high priority. We can go with the order restriction above at the beginning.

5.2 Representation of Families

There is one family for each $G \in \mathcal{G}$, hence the list of families is a (list of lists) of length `length(G)`, that is, the cardinality of $\mathcal{G}$. The i-th family is a model for the variables at nodes in `G[[i]]` given the variable at node `pred[i]` or given `root[i]` if `pred[i] == 0`.

(i) normal location-scale family with canonical statistics that are X and X^2, where X is the variable usually called normal, so the canonical parameters are $\theta_1 = \mu/\sigma^2$ and $\theta_2 = -1/2\sigma^2$ in terms of the usual parameters.

(j) gamma shape-scale family with canonical statistics that are X and $\log X$, where X is the variable usually called gamma, so the canonical parameters are $\theta_1 = \lambda$ and $\theta_2 = \alpha$ in terms of the usual parameters.

(k) multivariate Bernoulli with canonical statistics X_1, ..., X_k and parameter vector $\boldsymbol{\theta}_G = (\theta_1, \ldots, \theta_k)$ in terms of which the success probabilities (conditional mean value parameters are given by

$$\xi_j = \frac{e^{\theta_j}}{\sum_{i=1}^{k} e^{\theta_i}}$$

(l) multinomial with sample size n as a positive integer hyperparameter. If ψ_G is the cumulant function of the multivariate Bernoulli family, then we mean the family with cumulant function $n\psi_G$.

Constraints on parameter spaces for these families are discussed in Section 6 below.

Of the families discussed, all are identifiable except the multinomial families, (k) and (l), which are the subject of the following section.

For the multinomial families we do not need to specify the number of categories, the k in the descriptions in items (k) and (l) above, because it is the cardinality of the chain component $G \in \mathcal{G}$ with which the family is associated. This means that different chain components of different cardinality can have the same family specification `list(name = "multinomial")` and actually be families with different k because the cardinalities of the corresponding G are different. It is clear that the cardinality of G must be at least two for G to be associated with a multinomial model.

When the cardinality of G is two, and the parent variable $X_{p(G)}$ is also Bernoulli, we have the peculiar situation that the elements of the chain component are each Bernoulli and they satisfy $X_1 = X_{p(G)} - X_2$ (more on this issue in the following section), but this case cannot be replaced by the Bernoulli family, because it is a distribution for two nodes in the graph rather than one. Such a family serves as a "switch" activating either the descendants of X_1 or the descendants of X_2 but not both when $X_{p(G)} = 1$.

It is clear that the cardinality of G must be exactly two for G to be associated with a two-parameter normal or gamma model, items (i) and (j) in our model lists.

5.3 Dropping of Parameters for Multinomial

This is item 11 on our list. A multinomial conditional family for chain group G satisfies the data constraint

$$\sum_{j \in G} X_j = X_{p(G)}. \tag{3}$$

It follows that if we are fitting a conditional model that the vector $\mathbf{u}_G$ having components $u_{G,j}$ defined by

$$u_{G,j} = \begin{cases} 1, & j \in G \\ 0, & j \notin G \end{cases}$$

is a direction of constancy of the log likelihood, because

$$\langle \mathbf{X}, \boldsymbol{\theta} + s\mathbf{u}_G \rangle = \langle \mathbf{X}, \boldsymbol{\theta} \rangle + sX_{p(G)}$$

for any scalar s and we may drop the term $sX_{p(G)}$ from the log likelihood for this model, because $X_{p(G)}$ is not considered data for this model because it is conditional.

It follows that if we are fitting an unconditional model but $p(G)$ is a root node so $X_{p(G)}$ is constant, that exactly the same analysis holds and $\mathbf{u}_G$ is a direction of constancy of the log likelihood.

It follows that if we are fitting an unconditional model and $p(G)$ is not a root node that the vector $\mathbf{v}_G$ having components $v_{G,j}$ defined by

$$v_{G,j} = \begin{cases} 1, & j \in G \\ -1, & j = p(G) \\ 0, & \text{otherwise} \end{cases}$$

is a direction of constancy of the log likelihood, because

$$\langle \mathbf{X}, \boldsymbol{\varphi} + s\mathbf{v}_G \rangle = \langle \mathbf{X}, \boldsymbol{\varphi} \rangle$$

for any scalar s.

We should perhaps explain for those not familiar with the terminology that a *direction of constancy* of a log likelihood l is a vector $\mathbf{u}$ such that $l(\boldsymbol{\theta} + s\mathbf{u}) = l(\boldsymbol{\theta})$ for all s, just the situation we have in both cases above. Because the log likelihood of an exponential is strictly concave, a direction of constancy is the only form of non-identifiability an exponential family can have (which is why we use the concept here).

To unify our treatment of the cases let $\mathbf{w}_G$ be the direction of constancy, either $\mathbf{u}_G$ or $\mathbf{v}_G$ as the case may be. If the model matrix is $\mathbf{M}$, then we have non-identifiability if and only if $\mathbf{w}_G$ is in the column space of $\mathbf{M}$. This follows from the fact that the Fisher information matrix $\mathbf{I}(\boldsymbol{\eta})$ has a null eigenvector $\mathbf{w}$ if and only if $\mathbf{w}$ is a direction of constancy and the assumption that $\mathbf{M}$ has full rank, and that $\mathbf{M}^T\mathbf{I}(\boldsymbol{\eta})\mathbf{M}$ is the Fisher information for $\boldsymbol{\beta}$. (We concede this is not the way to write up a proof, but this is not a theory paper. We should write this up properly somewhere.)

However, it seems from this analysis that the following algorithm fixes the nonidentifiability problem.

- Add $\mathbf{w}_G$ as a column of $\mathbf{M}$. If there are multiple multinomial chain groups, add the $\mathbf{w}_G$ for each of them.

- Drop just enough columns of $\mathbf{M}$ to obtain a matrix of full rank, dropping none of the added $\mathbf{w}_G$.
- Then drop all of the added $\mathbf{w}_G$.

The resulting model matrix $\mathbf{M}$ is a maximal submatrix of the original $\mathbf{M}$ having the same row dimension subject to the conditions of (1) being full rank and (2) having no $\mathbf{w}_G$ in its column space.

All versions of the aster package already drop columns of the model matrix provided by the R function `model.matrix` or by the user, which is not always full rank. This code is in `aster.default` so it works no matter how the model matrix is provided. It uses the LINPACK QR decomposition routines, which provide rank estimation and some information about redundant columns in the pivoting information. This is not ideal. The exact arithmetic stuff in the `rcdd` package would do a perfect job, but we need to first upgrade the `rcdd` package to provide an interface to redundancy elimination and second fix the inability of `cddlib` to compile on Microsoft. But for the purposes of this section, how we do redundancy elimination is irrelevant. The point is that the aster package already must do redundancy elimination. This doesn't make the situation any worse than it already is. We merely have to add some additional redundancy to be eliminated.

5.4 Identifiability for Two-Parameter Normal and Gamma

As is obvious, a two-parameter model is not identifiable when the sample size is one. Thus we expect that the two-parameter normal and gamma models, items (i) and (j) on our model lists, are identifiable in the vast majority of applications, they need not be. Note that one chain group G with $X_{p(G)} = 1$ need not be a problem so long as there is more than one such group and the model matrix is sensible. An aster model in which every chain component is two-parameter normal and $p(G) = 1$ for all G generalizes ordinary least squares regression. If our model matrix is of the form

$$\begin{pmatrix} \mathbf{M} & \mathbf{0} \\ \mathbf{0} & \mathbf{I} \end{pmatrix}$$

where $\mathbf{M}$ is the what is usually thought of as the model matrix for regression, the upper block is for all of the first components of the chain groups ($\theta_1 = \mu/\sigma^2$, the lower block is for all of the second components ($\theta_2 = -1/2\sigma^2$), and $\mathbf{I}$ is the identity matrix, so we homoscedasticity, then everything is the same as in ordinary regression except that we take no account of error degrees of freedom (z-tests rather than t-tests, χ^2-tests rather than F-tests) and use

the MLE for $\hat{\sigma}^2$ dividing by n (the number of chain components) rather than error degrees of freedom.

I don't think the `lm` function in R worries about nonidentifiability (being unable to estimate the variance because $n = 1$). Perhaps we should, but it is not a high priority.

6 Constraints

This is item 8 on our list. It was partly dealt with in Section 3 above. The main point of this section is to mention all of the models introduced between here and Section 3 above that have constrained parameter spaces. All of the negative binomial models, which are items (b) and (c) in our model lists are one-parameter models with constraint $\theta < 0$. These were mentioned in Section 3. If we allow `list(name = "geometric")` as an abbreviation for `list(name = "negative.binomial", size = 1)` as was suggested in Section 4 in the discussion of user-level descriptions of families, then this is, of course, being a special case of the negative binomial, constrained in the same way.

The gamma scale family, which is item (h) in our model lists, is a one-parameter model with constraint $\theta > 0$.

The normal location-scale family, which is item (i) in our model lists, is a two-parameter model with constraint $\theta_2 < 0$ (and θ_1 unconstrained).

The gamma shape-scale family, which is item (j) in our model lists, is a two-parameter model with constraints $\theta_1 > 0$ and $\theta_2 > 0$.

All of these are regular exponential families, so the constraints do not need to be explicitly enforced if the optimization software can deal with with this situation, which was also discussed in Section 3 above (at the end of the section, on p. 5).

7 Validity Checks

This is item 3 on our list. There is a bug in all versions of aster to data (or a design misfeature) that the validity checks for aster models are not fine-grained enough. The problem is hard-wired. For an aster family as defined in `astfam.c` there are three functions you can call. One evaluates $\psi(\theta)$ or $\psi'(\theta)$, or $\psi''(\theta)$. Another checks data validity. The third simulates a realization from the model for a specified θ.

The particular problem we are on about here is the data validity check. When doing prediction of unconditional mean value parameters, we need

a data validity check that checks whether root data only is valid (because the prediction is a function of root data only). And this we do not have. The existing data validity check checks both child and parent data. In existing versions through the CRAN version we just set the child data to 1 because 1 is always valid for Bernoulli, Poisson, or zero-truncated Poisson. When we added two-truncated Poisson, this kludge gave errors (at least it gave a semi-understandable error rather than inexplicable crash). So we added an additional kludge to work around the lack of the needed validity check. We really need two validity check functions: one that checks the parent data for validity and another that checks that the child data (now possibly multivariate child data if the family is multivariate) given the parent data. Note that for exponential families data validity does not depend on parameter values.

Child data checks are not problematic. We always had them, and we continue to need them. Checks for new families that are brand name distributions are obvious, we just need to implement the checks when we implement everything else about the family. For less familiar families, the checks are also obvious from the definition. For k-truncated families, this implies $X > k$. For the two-parameter normal we must have $X^2 > 0$. This does not hold automatically, the user provides a two-dimensional canonical statistic, the first component of which purports to be $X_1 + \ldots + X_n$ where $n = X_{p(G)}$ and the second component of which purports to be $X_1^2 + \ldots + X_n^2$. We need to check that the latter is actually positive. We could further check that when $n = 1$ we actually have the first component the square of the second, or we could not bother to check.

One new data validity issue is mentioned in Section 1.2 of the revised aster paper (submitted to *Biometrika*). If the family is infinitely divisible, then parent data can be any nonnegative real value. If the family is not infinitely divisible, then parent data can be any nonnegative integer value. The items in our model lists that are infinitely divisible are item (b) negative binomial, item (d) Poisson, item item (g) normal-location, and item (h) gamma-scale.

Note that items (i) and (j), which one might think are infinitely divisible are not. For both the distribution of X is infinitely divisible, but the canonical statistics for these families, which are vectors, (X, X^2) for one and $(X, \log X)$ for the other, do not have infinitely divisible distributions. Curiously for the normal, both marginal distributions, for X and X^2, are infinitely divisible, but the joint distribution of (X, X^2) is not.

No version to date had a parameter validity issue. All models were one-dimensional and their canonical parameter spaces were the whole real line.

Now we have constraints. As discussed at the end of Section 3 (p. 5 above) when evaluating ψ_G, $\nabla\psi_G$, and $\nabla^2\psi_G$ all parameter values are "valid", even those outside the canonical parameter space, we merely need to return `Inf` for ψ_G and `NaN` for components of the derivatives. The optimization software needs to be smart enough to deal with this by attempting a smaller step to some point that is in the parameter space. (Whether this leads to code changes other than in the function that evaluates the cumulant function and its derivatives, I do not know. Presumably no other code changes are necessary, but there may be some assumptions of finiteness somewhere in the code that I forgot.)

The simulation function does need to check for valid parameter values and crash with an informative error message if not. So perhaps we should provide a parameter validity check function, since we need to do the validity check in two different bits of code (cumulant and simulation) anyway.

8 Mixed Parameterization

This is item 9 on our list. I puzzled for a long time about whether mixed parameterizations make sense. Barndorff-Nielsen (*Information and Exponential Families*, 1978) talks about mixed parameterizations in which some parameters are canonical and some mean-value. That is not what I mean. What I mean here is when some canonical parameters are conditional θ and some are unconditional φ.

It is in fact obvious from the discussion the "key" equation of aster theory, equation (5) in the revised aster manuscript, that if we pick an arbitrary subset of nodes to make unconditional so their linear predictor is $\eta_j = \varphi_j$ given by the "key" equation and leave the rest of the nodes with $\eta_j = \theta_j$ as linear predictor, that if we want to find the $\boldsymbol{\theta}$ corresponding to a $\boldsymbol{\eta}$ that so long as we solve for children before parents that this solving merely moves a term from one side of the "key" equation to the other for those j such that $\eta_j = \varphi_j$ and otherwise does nothing.

So mixed parameterizations are possible. Whether they are worth implementing, whether they would be scientifically interesting for any application, I do not know.

9 MLE of Non-Exponential Family Parameters

This is item 12 on our list. Everything discussed above does not change the basic notion of aster models that all parameters are exponential family

parameters. Hyperparameters are not considered parameters. They are "assumed known."

If we wanted to maximize over a hyperparameter, the log likelihood calculated by `mlogl` would be of no help because we drop terms not containing what we consider to be the parameters (that is, the exponential family parameters $\boldsymbol{\theta}$). Since we drop terms containing the hyperparameters, the log likelihood we compute is not the log likelihood when the hyperparameters are considered parameters.

We could, as a kludge, supposing, for example, we wanted to fit the size parameter of a negative binomial distribution by maximum likelihood use the aster model machinery to fit the aster MLE for each value of size parameter α on a grid of values, then evaluate the log likelihood correctly (not using the aster package, using `dnbinom` and perhaps other functions), and then maximize this one-parameter profile. But ideally we should not have to kludge this. The aster package should handle this too. But that would require a lot of changes to the machinery and is not a high priority.

People who like homoscedasticity assumptions would prefer the one-parameter normal location model, item (g) in our model lists, to the two-parameter normal model, item (i) in our model lists, if the hyperparameter σ in the one-parameter family could participate in maximum likelihood, so that would be one reason to add this feature.

10 User-Specified Superfamilies

This is item 10 on our list. Now that we have the distinction between families and superfamilies, introduced on p. 6 above. We can see that what we want to be "user-specified" by writing an implementation in R (to be called from inside the aster C implementation) is a superfamily. The user has the same requirements that implementers of superfamilies (in C) now have. A superfamily has the following tasks

- Calculation of ψ, ψ', and ψ'' for a one-parameter family or ψ, $\nabla\psi$, and $\nabla^2\psi$ for a multi-parameter family.

  Note that for either we need all of the hyperparameter values, and for the latter we also need the dimension (the cardinality of the chain component). These are provided in the function call.

- Data validity checks. One function checks validity of parent data. Another checks validity of both parent and child data. (Or the same

function does both jobs, as in current implementations, but there is a flag that says check parent data only.)

- Simulation. A function that simulates one realization from the conditional distribution specified by the family (hyperparameters, parameters, and dimension given).

- The superfamily must say what the maximum and minimum values for the cardinality of a chain component are: both equal to one for an inherently one-parameter family, both equal to two for an inherently two-parameter family, two to infinity for a multinomial family.

- The superfamily must say what the number of hyperparameters are.

All of these will need to be specified by R function (for the first three) or by R data (for the last two).

References

Geyer, C. J., Wagenius, S., and Shaw, R. G. (2005). Aster models for life history analysis. University of Minnesota School of Statistics, Technical Report 644. `http://www.stat.umn.edu/geyer/aster/aster.pdf` submitted August 2005. `http://www.stat.umn.edu/geyer/aster/aster-2a.pdf` revised and resubmitted June 2006.