Aster Models
Stat 8053 Lecture Notes

Charles J. Geyer

School of Statistics
University of Minnesota

October 27, 2014

Aster Models

Aster models (named after the flowers)

Charles J. Geyer, Stuart Wagenius, and Ruth G. Shaw (2007).
Aster Models for Life History Analysis.
Biometrika, 94, 415–426.

are a new kind of exponential family regression model (canonical affine submodels of regular full exponential families) that allow for

- dependence among components of the response vector, which is specified by a graphical model, and
- components of the response vector having different familes, some Bernoulli, some Poisson, some zero-truncated Poisson, some normal, etc.

Aster Models (cont.)

The main point of these slides **is not** to get you to fully understand aster models. That would take all semester and was done last fall in a special topics course. All of the slides for that course and recorded sound from the lectures are at

`http://users.stat.umn.edu/geyer/8931aster/`

The main point of these slides **is** to get you to have a vague understanding of aster models, enough to get the point of how powerful exponential family regression models can be.

Aster Models in R

R contributed package `aster` on CRAN.

```
install.packages("aster")
library(aster)
```

Function `aster` fits models. Generic functions `summary`, `predict`, and `anova` work like those for linear and generalized linear models.

Aster Models on the Web

Main aster web page

`http://www.stat.umn.edu/geyer/aster/`

has links to papers and tech reports. All tech reports done with `Sweave` so everything is exactly reproducible.

Google group

`https://groups.google.com/forum/#!forum/aster-analysis-user-group`

Aster Models (cont.)

Lots of papers.

I am co-author on 5.

My sister, Ruth Shaw, Professor in the Department of Ecology, Evolution, and Behavior on the St. Paul Campus, is a co-author of those and on several more.

Dan Eck is lead author on yet another (almost ready to submit).

Dozens of papers by biologists not in our group.

Life History Analysis

Life history analysis (LHA) follows organisms over the course of their lives collecting various data: survival through various time periods and also various other data, which only makes sense conditional on survival.

Thus LHA generalizes **survival analysis**, which only uses data on survival.

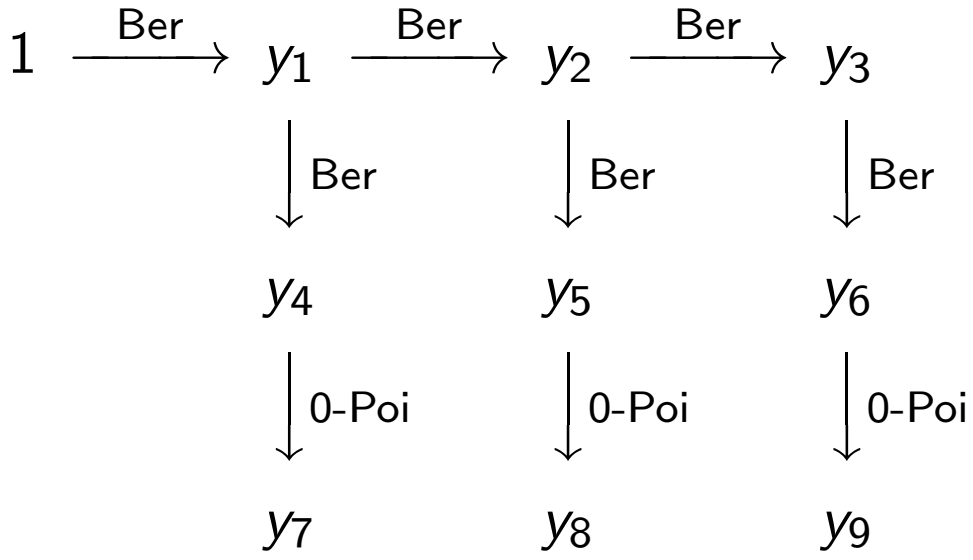
The LHA of interest to many biologists concerns **Darwinian fitness** conceptualized as the lifetime number of offspring an organism has. The various bits of data collected over the course of the life that contribute to this are called **components of fitness**.

Life History Analysis (cont.)

The fundamental statistical problem of LHA is that overall fitness, considered as a random variable, fits (in the statistical sense) no brand-name distribution. It has a large atom at zero (individuals that died without producing offspring) as well as multiple modes (one for each breeding season the organism survives). No statistical methodology before aster deals with data like that.

This issue has long been well understood in the LHA literature. So what was done instead was analyze components of fitness separately conditional on survival, but this doesn't address the variable (overall fitness) of primary interest (an issue also well understood, but you do what you can do).

An Aster Graph



y_i are components of response vector for one individual (all individuals have isomorphic graphs). 1 is the constant 1.

Arrows indicate conditional distributions of variable at head of arrow (successor) given variable at tail of arrow (predecessor). Ber = Bernoulli, 0-Poi = zero-truncated Poisson.

Graphical Terminology

For one arrow in a graph

$$y_2 \xrightarrow{\text{Ber}} y_3$$

we say y_3 is the *successor* of y_2 and (conversely) we say y_2 is the *predecessor* of y_3.

A node of the graph (random variable) having no successors is called a *terminal node* of the graph.

A node of the graph (random variable) having no predecessors is called an *initial node* of the graph.

General Aster Graphs

Graphs for aster models have the following properties

- they are acyclic (there is no path following arrows in the directions they point that gets back to where it started)
- every node has at most one predecessor (initial nodes have none, non-initial nodes have one),
- arrows represent conditional distributions of successor given predecessor, and
- each such distribution is one-parameter exponential family with
  - the successor is the canonical statistic and
  - the predecessor is the sample size (more on this presently).

Aster Model Joint Distribution

Nodes (variables) have at most one predecessor, hence graph is specified by function p that maps from set J of non-initial nodes to set N of all nodes. $y_{p(j)}$ is predecessor of y_j.

y_j at initial nodes treated as constants. Then, because graph is acyclic, joint distribution factors as product of conditionals

$$f_\theta(y) = \prod_{j \in J} f_\theta(y_j \mid y_{p(j)})$$

Log likelihood is

$$l(\theta) = \sum_{j \in J} \log f_\theta(y_j \mid y_{p(j)})$$

Aster Model Joint Distribution (cont.)

$$f_\theta(y) = \prod_{j \in J} f_\theta(y_j \mid y_{p(j)})$$

In a term $f_\theta(y_j \mid y_{p(j)})$ where $y_{p(j)}$ is an initial node, hence a constant random variable, conditioning on a constant random variable is like not conditioning at all

$$f_\theta(y_j \mid y_{p(j)}) = \frac{f_\theta(y_j, y_{p(j)})}{f_\theta(y_{p(j)})} = f_\theta(y_j)$$

because $f_\theta(y_{p(j)}) = 1$ and $f_\theta(y_j, y_{p(j)}) = f_\theta(y_j)$ when $y_{p(j)}$ has the only value it is possible for it to have (the constant it is).

Predecessor is Sample Size

An arrow

$$y_{p(j)} \xrightarrow{\text{whatever}} y_j$$

indicates that the conditional distribution of y_j given $y_{p(j)}$ is the distribution of the sum of IID (independent and identically distributed) random variables having the "whatever" distribution, and there are $y_{p(j)}$ terms in the sum.

By convention, a sum with zero terms is zero.

Hence the conditional distribution of y_j given $y_{p(j)}$ is

- concentrated at zero when $y_{p(j)} = 0$,
- is the "whatever" distribution when $y_{p(j)} = 1$, and
- is the sum of k IID "whatever" distributed random variables when $y_{p(j)} = k$.

Predecessor is Sample Size (cont.)

$$1 \xrightarrow{\text{Ber}} y_1 \xrightarrow{\text{Whatever}} y_2$$

The unconditional distribution of y_1 is Bernoulli, hence zero-or-one-valued.

The conditional distribution of y_2 given y_1 is

- degenerate, concentrated at zero if $y_1 = 0$
- Whatever if $y_1 = 1$

We see that, in the zero-or-one-valued predecessor case, the interpretation is simple. Predecessor $= 0$ implies successor $= 0$. Otherwise, the conditional distribution of the successor is the named distribution.

Predecessor is Sample Size (cont.)

But predecessors do not have to be zero-or-one-valued.

$$1 \xrightarrow{\text{Poi}} y_1 \xrightarrow{\text{Ber}} y_2$$

Now y_1 is nonnegative-integer-valued (Poi = Poisson).

The sum of n IID Bernoulli random variables is binomial with sample size n.

The conditional distribution of y_2 given y_1 is

- degenerate, concentrated at zero if $y_1 = 0$
- Binomial with sample size y_1 if $y_1 > 0$

The Zero-Truncated Poisson Distribution

Zero-truncated Poisson is a Poisson random variable conditioned on not being zero. The probability mass function (PMF) is

$$f(x) = \frac{\mu^x e^{-\mu}}{x!(1 - e^{-\mu})}, \qquad x = 1, 2, \ldots,$$

where $\mu > 0$ is the mean of the untruncated Poisson variable, (just the Poisson PMF divided by the probability the Poisson variable is nonzero, which is $1 - e^{-\mu}$).

Zero-Truncated and Zero-Inflated

The reason why we want the zero-truncated Poisson distribution is that sometimes random variables are zero for reasons other than Poisson variation.

If we want to deal with this we need the so-called zero-inflated Poisson distribution (about which there has been much recent literature, nearly 5,000 hits in Google Scholar).

Zero-Truncated and Zero-Inflated (cont.)

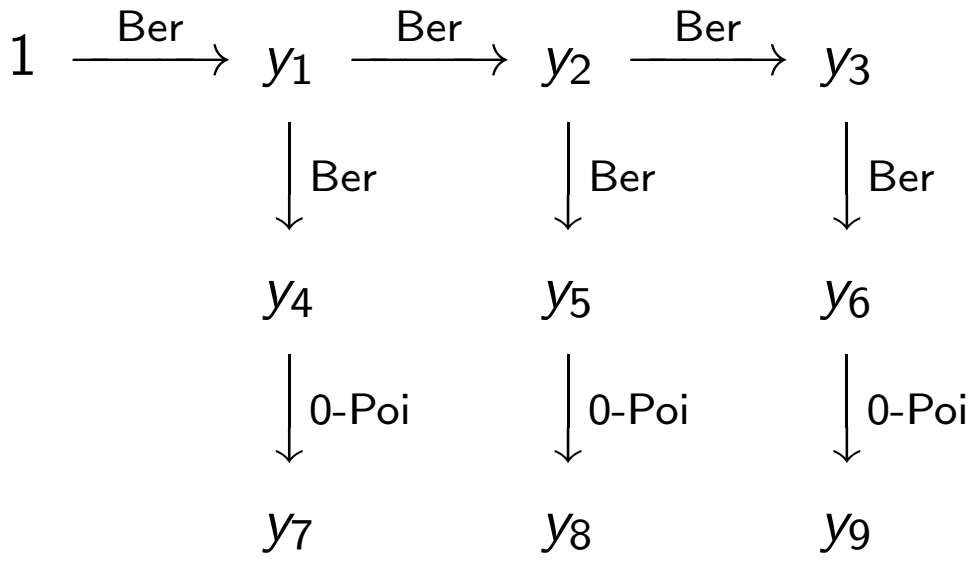
Because aster models have one parameter per arrow, the aster model way to get the zero-inflated Poisson distribution uses two arrows rather than one

$$y_i \xrightarrow{\text{Ber}} y_j \xrightarrow{\text{0-Poi}} y_k$$

The conditional distribution of y_k given y_i (both arrows) is

- degenerate, concentrated at zero if $y_i = 0$
- zero-inflated Poisson, if $y_i = 1$
- the sum of y_i IID zero-inflated Poisson random variables, if $y_i > 1$

An Aster Graph (cont.)



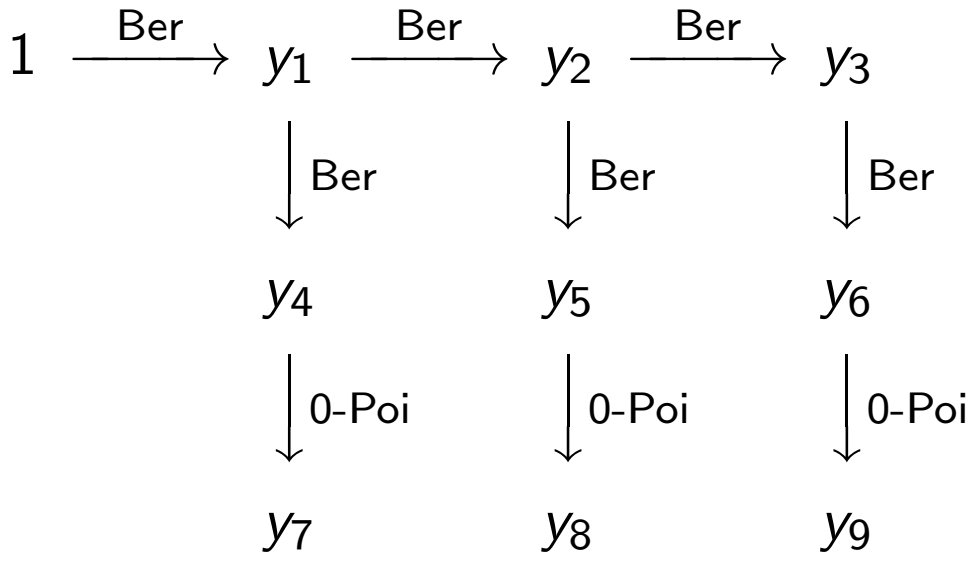
Graph for *Echinacea angustifolia* example in Geyer, Wagenius and Shaw (*Biometrika*, 2007).

y_1, y_2, y_3 indicate survival in each of three years (2002–2004).

y_4, y_5, y_6 indicate flowering status (1 = some flowers, 0 = no flowers) in corresponding years.

y_7, y_8, y_9 are flower counts in corresponding years.

An Aster Graph (cont.)



The top layer (survival indicators) are necessary to model survival components of fitness.

The middle and bottom layers (flowering indicators and flower counts) are necessary to model fecundity components of fitness while accounting for zero-inflation of the Poisson distributions.

Aster Model Log Likelihood

$$l(\theta) = \sum_{j \in J} \log f_\theta(y_j \mid y_{p(j)})$$

Because of the way IID works for exponential families (Section 1.13 of the handout on exponential families) and because each arrow corresponds to a one-parameter exponential family with y_j as canonical statistic and $y_{p(j)}$ as sample size,

$$\log f_\theta(y_j \mid y_{p(j)}) = y_j \theta_j - y_{p(j)} c_j(\theta_j)$$

here θ_j is the canonical parameter and c_j is the cumulant function (for this one-parameter exponential family).

Aster Model Log Likelihood (cont.)

$$l(\theta) = \sum_{j \in J} \left[y_j \theta_j - y_{p(j)} c_j(\theta_j) \right]$$

$$= \sum_{j \in J} y_j \left[\theta_j - \sum_{\substack{k \in J \\ j = p(k)}} c_k(\theta_k) \right] - \sum_{\substack{k \in J \\ p(k) \notin J}} y_{p(k)} c_k(\theta_k)$$

This is recognizable as log likelihood for joint exponential family.
Blue term is j-th component of joint canonical parameter vector.
Red term is cumulant function of joint family.

Aster Model Log Likelihood (cont.)

$$l(\varphi) = \langle y, \varphi \rangle - c(\varphi)$$

where

$$\varphi_j = \theta_j - \sum_{\substack{k \in J \\ j = p(k)}} c_k(\theta_k), \qquad j \in J$$

and

$$c(\varphi) = \sum_{\substack{k \in J \\ p(k) \notin J}} y_{p(k)} c_k(\theta_k)$$

Aster Transform

Map between θ and φ is invertible

$$\theta_j = \varphi_j + \sum_{\substack{k \in J \\ j = p(k)}} c_k(\theta_k)$$

where θ_k on right-hand side have "already" been determined as function of φ. Use in any order that does successors before predecessors (always is one because graph is acyclic).

Aster Transform (cont.)

We call θ the *conditional canonical parameter vector*.
We call φ the *unconditional canonical parameter vector*.

They are not as parallel as the names suggest.

The vector θ has components that are the canonical parameters of the conditional distributions associated with the arrows of the graph.

The vector φ is the canonical parameter vector of the joint distribution of the aster model (which is an exponential family).

The Aster Transform (cont.)

Each θ_j is the canonical parameter of a one-parameter exponential family model (for one arrow). The vector θ is not a canonical parameter vector of a multivariate exponential family.

The vector φ is the canonical parameter vector of a multivariate exponential family. Each φ_j is not a canonical parameter of a one-parameter exponential family.

The Aster Transform (cont.)

Are you lost? If so, no surprise.

The aster transform makes mathematical-statistical-theoretical sense, but it doesn't make common sense. It is not intuitive at all.

To understand it we must apply Zen and not try to understand it.

If that doesn't make sense, wait a while. We hope you will eventually achieve enlightenment.

The technical report *A Philosophical Look at Aster Models* goes through one very simple example, but it only shows the algebraic formulas are a big mess that no one can understand intuitively. (The whole point of the example is to show you that you do not want to try to understand the aster transform by staring at the formulas.)

The Aster Transform (cont.)

A quote from my master's level theory notes

> *Parameters are meaningless quantities. Only probabilities and expectations are meaningful.*

Of course, some parameters are probabilities and expectations, but most exponential family canonical parameters are not.

A quote from *Alice in Wonderland*

> *'If there's no meaning in it,' said the King, 'that saves a world of trouble, you know, as we needn't try to find any.'*

Realizing that canonical parameters are meaningless quantities "saves a world of trouble". We "needn't try to find any".

Mean Value Parameters

Define

$$\xi_j = E\{y_j \mid y_{p(j)} = 1\}$$
$$\mu_j = E(y_j)$$

By properties of exponential families

$$\xi_j = c_j'(\theta_j)$$
$$\mu = \nabla c(\varphi)$$

where prime denotes univariate derivative and ∇ denotes multivariate derivative (vector of partial derivatives).

By properties of exponential families these changes of parameters are also invertible (although no closed-form expression in general).

Mean Value Parameterizations (cont.)

It is useful to examine the direct change of parameter

$$\mu \longleftrightarrow \xi$$

rather than the long way round

$$\mu \longleftrightarrow \varphi \longleftrightarrow \theta \longleftrightarrow \xi$$

Applying the iterated expectation theorem to

$$E(y_j|y_{p(j)}) = y_{p(j)}\xi_j$$

gives

$$\mu_j = E(y_j) = E\{E(y_j|y_{p(j)})\} = E(y_{p(j)}\xi_j) = \xi_j E(y_{p(j)}) = \xi_j \mu_{p(j)}$$

Mean Value Parameterizations (cont.)

And iterating this gives

$$\begin{aligned}\mu_j &= \xi_j \mu_{p(j)} \\ &= \xi_j \xi_{p(j)} \mu_{p(p(j))} \\ &= \xi_j \xi_{p(j)} \xi_{p(p(j))} \mu_{p(p(p(j)))} \\ &= \xi_j \xi_{p(j)} \xi_{p(p(j))} \xi_{p(p(p(j)))} \mu_{p(p(p(p(j))))}\end{aligned}$$

and so forth.

Keep going until the only μ is for an initial node, in which case, since the expectation of a constant is a constant,

$$\mu_{p(p(p(p(j))))} = y_{p(p(p(p(j))))}$$

(or perhaps with more p's, whatever it takes to get to an initial node).

Mean Value Parameterizations (cont.)

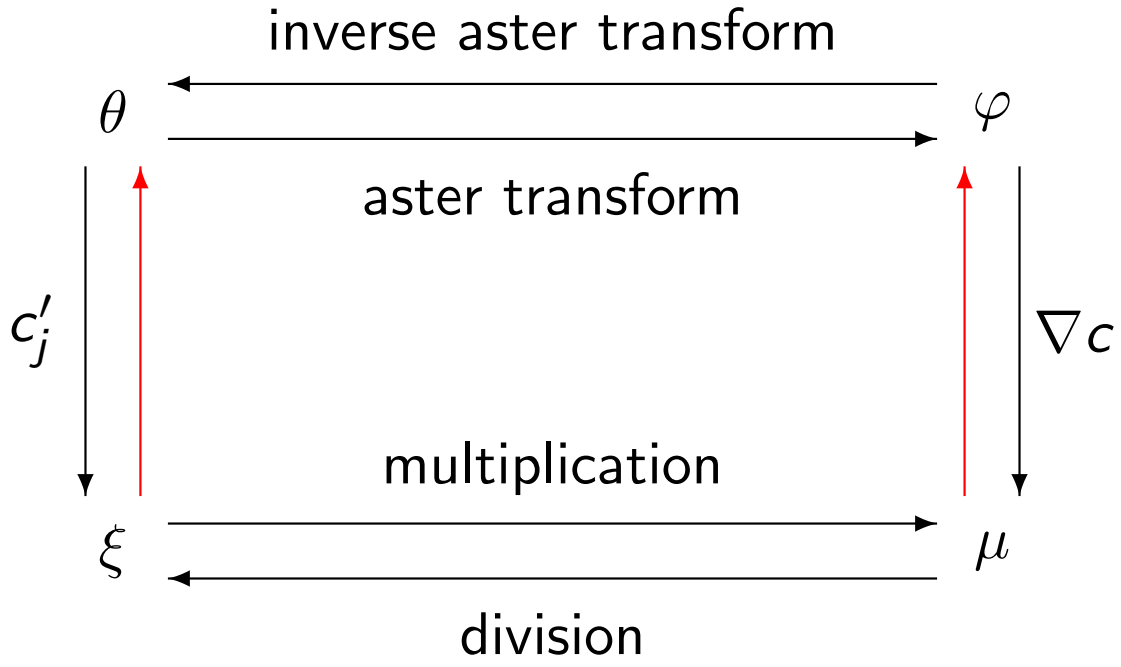
Going the other way is even easier

$$\xi_j = \frac{\mu_j}{\mu_{p(j)}}$$

assuming we do not have divide by zero. Since we already know that the mapping $\mu \longleftrightarrow \xi$ is invertible, it must be that we never have divide by zero (this follows from the aster model distribution being non-degenerate).

A Plethora of Parameterizations

Now we have four different parameterizations. All are equally good, and any one can be mapped to any other.



(no closed-form expression for red arrows).

A Plethora of Parameters

Four different parameterizations μ, ξ, θ, and φ.

All are important. All play a role in some scientific arguments.
Users have to understand all four.

But wait, there's more!

Canonical Affine Submodels

In an exponential family, with canonical parameter φ, the change of parameter

$$\varphi = a + M\beta$$

where a is a known vector, called the offset vector, and M is a known matrix, called the model matrix, gives a new exponential family (Section 1.15 of the handout on exponential families)

Submodel canonical parameter vector is β.
Submodel canonical statistic vector is $M^T y$.
Submodel mean value parameter vector is $\tau = E(M^T y) = M^T \mu$.

A Plethora of Parameters (cont.)

Six different parameterizations

μ	saturated model	unconditional	mean value
ξ	saturated model	conditional	mean value
φ	saturated model	unconditional	canonical
θ	saturated model	conditional	canonical
β	submodel	unconditional	canonical
τ	submodel	unconditional	mean value

Fisher Information

Fisher information for submodel canonical parameter vector β is

$$I(\beta) = -\nabla^2 l(\beta) = M^T \nabla^2 c(M\beta) M$$

Computer can convert to any other parameterization. And compute derivatives for applying the delta method to transfer standard errors.

Interpretation

The pillars of interpretation of an aster model, just like for any exponential family model, are

- sufficient dimension reduction (submodel canonical statistic is sufficient, Section 1.17 of exponential family handout),
- observed equals expected (MLE of submodel mean value parameter equals submodel sufficient statistic, Section 1.12.2 of exponential family handout),
- maximum entropy (submodel leaves every aspect of data as random as possible given that it fixes the submodel mean value parameter, Section 1.18 of exponential family handout), and
- multivariate monotonicity of the map $\varphi \longleftrightarrow \mu$ and univariate monotonicity of the map $\theta_j \longleftrightarrow \xi_j$ (Section 1.11 of exponential family handout).

Example Data Analysis

Data are found in the R dataset `echinacea` in the R package aster.

```
> library(aster)
> data(echinacea)
> class(echinacea)

[1] "data.frame"

> dim(echinacea)

[1] 570  12

> names(echinacea)

 [1] "hdct02" "hdct03" "hdct04" "pop"    "ewloc"
 [6] "nsloc"  "ld02"   "fl02"   "ld03"   "fl03"
[11] "ld04"   "fl04"
```

Example Data Analysis (cont.)

The variables that correspond to nodes of the graph are, in the order they are numbered in the graph

```
> vars <- c("ld02", "ld03", "ld04", "fl02", "fl03",
+     "fl04", "hdct02", "hdct03", "hdct04")
```

The graphical structure is specified by a vector that gives for each node the index (not the name) of the predecessor node or zero if the predecessor is an initial node.

```
> pred <- c(0, 1, 2, 1, 2, 3, 4, 5, 6)
```

This says the first node given by the vars vector is initial (because `pred[1] == 0`), the predecessor of the second node given by the vars vector is the first node given by the vars vector (because `pred[2] == 1`), and so forth.

Example Data Analysis (cont.)

Let's check this makes sense.

```
> foo <- rbind(vars, c("initial", vars)[pred + 1])
> rownames(foo) <- c("successor", "predecessor")
> foo
```

```
             [,1]      [,2]   [,3]   [,4]   [,5]
successor   "ld02"    "ld03" "ld04" "fl02" "fl03"
predecessor "initial" "ld02" "ld03" "ld02" "ld03"
             [,6]   [,7]     [,8]     [,9]
successor   "fl04" "hdct02" "hdct03" "hdct04"
predecessor "ld04" "fl02"   "fl03"   "fl04"
```

That's right.

Example Data Analysis (cont.)

The last part of the specification of the graph is given by a corresponding vector of integers coding families (distributions). The default is to use the codes: 1 = Bernoulli, 2 = Poisson, 3 = zero-truncated Poisson. Optionally, the integer codes specify families given by an optional argument `famlist` to functions in the `aster` package, and this can specify other distributions besides those in the default coding.

```
> fam <- c(1, 1, 1, 1, 1, 1, 3, 3, 3)
> rbind(vars, fam)
```

```
     [,1]   [,2]   [,3]   [,4]   [,5]   [,6]
vars "ld02" "ld03" "ld04" "fl02" "fl03" "fl04"
fam  "1"    "1"    "1"    "1"    "1"    "1"
     [,7]     [,8]     [,9]
vars "hdct02" "hdct03" "hdct04"
fam  "3"      "3"      "3"
```

Example Data Analysis (cont.)

There is one more step before we can fit models. The R function `aster` which fits aster models wants the data in "long" rather than "wide" format, the former having one line per node of the graph rather than one per individual.

The magic incantation to do this is

```
> redata <- reshape(echinacea, varying = list(vars),
+     direction = "long", timevar = "varb",
+     times = as.factor(vars), v.names = "resp")
> redata <- data.frame(redata, root = 1)
```

If you forget this incantation, it and everything else we have done in this example is on the help page for the R function `aster` obtained by doing

```
help(aster)
```

Example Data Analysis (cont.)

```
> class(redata)

[1] "data.frame"

> dim(redata)

[1] 5130    7

> sapply(redata, class)

      pop      ewloc      nsloc       varb       resp
 "factor" "integer" "integer"  "factor" "integer"
       id       root
"integer" "numeric"
```

Example Data Analysis (cont.)

```
> names(redata)
```

```
[1] "pop"   "ewloc" "nsloc" "varb"  "resp"  "id"
[7] "root"
```

All of the variables in `echinacea` that are named in `vars` are gone. They are packed into the variable `resp`. Which components of `resp` correspond to which components of `vars` is shown by the new variable `varb`

```
> levels(redata$varb)
```

```
[1] "fl02"   "fl03"   "fl04"   "hdct02" "hdct03"
[6] "hdct04" "ld02"   "ld03"   "ld04"
```

Example Data Analysis (cont.)

Now we have all of the response variables (components of fitness) collected into a single vector `resp` and we have learned what `varb` is. What about the other variables?

`root` we defined ourselves. When the predecessor of a node is initial, then the corresponding component of `root` gives the value of the predecessor. Other components of `root` are ignored. We set them all to one.

`id` is seldom (if ever) used. It tells what individual (what row of the original data frame `echinacea`) a row of `reshape` came from.

`nsloc` (north-south location) and `ewloc` (east-west location) give the position each individual was located in the experimental plot.

Example Data Analysis (cont.)

pop gives the ancestral populations: each individual was grown from seed taken from a surviving population in a prairie remnant in western Minnesota near the Echinacea Project field site.

```
> levels(redata$pop)
```

```
[1] "AA"       "Eriley"   "Lf"       "Nessman" "NWLF"
[6] "SPP"      "Stevens"
```

Regression Models

Different families for different nodes of the graph means it makes no sense to have terms of the regression formula applying to different nodes.

In particular, it makes no sense to have one "intercept" for all nodes.

To in effect get a different "intercept" for each node in the graph, include `varb` in the formula

```
y ~ varb + ...
```

The categorical variable `varb` gets turned into as many dummy variables as there are nodes in the graph, one is dropped, and the "intercept" dummy variable (all components $= 1$) is added; the effect is to provide a different intercept for each node.

Technical Quibble

Why is does the variable named `varb` have that name?

Because of the optional argument `timevar = "varb"` supplied to the "magic incantation" (`reshape` function)

```
redata <- reshape(echinacea, varying = list(vars),
    direction = "long", timevar = "varb",
    times = as.factor(vars), v.names = "resp")
```

We could have given it the name `fred` or `sally` or whatever we want. I picked `varb` (short for "variables") without thinking about it the first time I did this, and everyone (including me) has just copied that ever since.

Similarly the name `resp` for the response is specified by the optional argument `v.names = "resp"`.

Regression Models (cont.)

Similar thinking says we want completely different regression coefficients of all kinds of predictors for each node of the graph. That would lead us to formulas like

```
y ~ varb + varb : (...)
```

where ... is any other part of the formula.

The `:` operator in the R formula mini-language denotes interactions without main effects. The `*` operator in the R formula mini-language denotes interactions with main effects. That is, `a * b` means the same thing as `a + b + a : b`

So the above formula says we want the "main effects" for `varb`, and we want the "interaction" of `varb` with "everything else" (the ...), but we do not want the "main effects" for "everything else".

Regression Models (cont.)

```
y ~ varb + varb : (...)
```

Having said that, we immediately want to take it back. The language of "main effects" and "interactions" was never designed to apply to aster models.

We should not think of this formula as specifying "main effects" for `varb` (whatever that may mean) but rather as specifying a separate "intercept" for each node of the graph.

Similarly, we should not think of this formula as specifying "interaction" between `varb` and "everything else" (whatever that may mean) but rather as specifying separate coefficients for "everything else" for each node of the graph.

Regression Models (cont.)

Thus IMHO (in my humble opinion) you should always say "main effects" and "interactions" with scare quotes, emphasizing that these terms are at best highly misleading and confusing.

Regression Models (cont.)

```
y ~ varb + varb : (...)
```

But formulas like this would yield too many regression coefficients to estimate well! We can do better!

Maybe we don't really need different regression coefficients for each node. Maybe different for each kind of node (whatever that may mean) would be enough.

Regression Models (cont.)

```
> layer <- gsub("[0-9]", "", as.character(redata$varb))
> unique(layer)
```

```
[1] "ld"   "fl"   "hdct"
```

```
> redata <- data.frame(redata, layer = layer)
> with(redata, class(layer))
```

```
[1] "factor"
```

Maybe

```
y ~ varb + layer : (...)
```

good enough?

Regression Models (cont.)

```
y ~ varb + layer : (...)
```

But formulas like this would still yield too many regression coefficients to estimate well! We can do better!

Because of the way the aster transform works regression coefficients "for" a node of the graph also influence all "earlier" nodes of the graph (predecessor, predecessor of predecessor, predecessor of predecessor of predecessor, etc.)

So maybe it would be good enough to only have separate coefficients for the "layer" of the graph consisting of terminal nodes?

Regression Models (cont.)

```
> fit <- as.numeric(layer == "hdct")
> unique(fit)
```

```
[1] 0 1
```

```
> redata <- data.frame(redata, fit = fit)
> with(redata, class(fit))
```

```
[1] "numeric"
```

Maybe

```
y ~ varb + fit : (...)
```

good enough?

Regression Models (cont.)

We called this variable we just made up `fit`, short for Darwinian fitness.

With formulas like

```
y ~ varb + fit : (...)
```

the regression coefficients in terms specified by ... have a direct relationship with expected Darwinian fitness. And that's usually what is wanted in LHA.

A Technical Quibble

We shouldn't have said Darwinian fitness. Rather we shouldn't have said *the best surrogate of Darwinian fitness in these data*.

Flower counts are not "lifetime number of offspring". Still less are flower counts over three years (not the whole life span).

Other *Echinacea* data (Wagenius, et al., 2010, *Evolution*) have more years and more components of fitness.

Other data on other species (Stanton-Geddes, et al., 2012, *PLoS One*) have "best surrogate of fitness" pretty close to "fitness" (with no qualifiers).

After we have emitted academic weasel-wording making clear that we are aware of the difference between what we are calling fitness and the Platonic ideal of fitness, we can just drop the fuss and go on with the analysis and interpretation.

Regression Models (cont.)

In practice we use formulas like

```
y ~ varb + layer : (...) + fit : (...)
```

with the two ... having different formula terms.

The formula terms in the second ... are the ones that we want to say have a direct effect on fitness (and want statistics to tell us whether they do or not).

The formula terms in the first ... are everything else (the terms whose effect on fitness, if any, is not an issue of scientific interest in this experiment).

No Naked Predictors

We summarize our advice about formulas for aster models with the slogan

> *No naked predictors!*

or more precisely

> *No naked predictors except* `varb` *and factor or indicator variables derived from it, like* `layer` *and* `fit`

Our slogan means every predictor other than these must occur "interacted with" one of these.

Example Data Analysis (cont.)

```
> aout <- aster(resp ~ varb + layer : (nsloc + ewloc) +
+     fit : pop, pred, fam, varb, id, root, data = redata)
> summary(aout)

Call:
aster.formula(formula = resp ~ varb + layer:(nsloc + ewloc)
    fit:pop, pred = pred, fam = fam, varvar = varb, idvar =
    root = root, data = redata)

                Estimate Std. Error z value Pr(>|z|)
(Intercept)    -1.050644   0.184332  -5.700 1.20e-08
varbfl03       -0.349096   0.267919  -1.303   0.1926
varbfl04       -0.344222   0.243899  -1.411   0.1581
varbhdct02      1.321414   0.261174   5.060 4.20e-07
varbhdct03      1.343374   0.214625   6.259 3.87e-10
varbhdct04      1.851328   0.199853   9.263  < 2e-16
varbld02       -0.029302   0.315703  -0.093   0.9260
```

Example Data Analysis (cont.)

	Estimate	Std. Error	z value	Pr(>|z|)
(Intercept)	-1.0506435	0.1843320	-5.6997	1.200e-08
varbfl03	-0.3490958	0.2679185	-1.3030	0.19258
varbfl04	-0.3442222	0.2438992	-1.4113	0.15815
varbhdct02	1.3214136	0.2611741	5.0595	4.203e-07
varbhdct03	1.3433740	0.2146250	6.2592	3.870e-10
varbhdct04	1.8513276	0.1998528	9.2635	< 2.2e-16
varbld02	-0.0293022	0.3157033	-0.0928	0.92605
varbld03	1.7400507	0.3961890	4.3920	1.123e-05
varbld04	4.1885771	0.3342661	12.5307	< 2.2e-16
layerfl:nsloc	0.0701024	0.0146520	4.7845	1.714e-06
layerhdct:nsloc	-0.0058043	0.0055499	-1.0458	0.29564
layerld:nsloc	0.0071652	0.0058667	1.2213	0.22196
layerfl:ewloc	0.0179769	0.0144128	1.2473	0.21229
layerhdct:ewloc	0.0076060	0.0055608	1.3678	0.17138
layerld:ewloc	-0.0047874	0.0059191	-0.8088	0.41863
fit:popAA	0.1292377	0.0891292	1.4500	0.14706
fit:popEriley	-0.0495612	0.0712789	-0.6953	0.48686
fit:popLf	-0.0332786	0.0795727	-0.4182	0.67579
fit:popNessman	-0.1862690	0.1277869	-1.4577	0.14494
fit:popNWLF	0.0210283	0.0635998	0.3306	0.74092
fit:popSPP	0.1491795	0.0677156	2.2030	0.02759

Example Data Analysis (cont.)

The regression coefficients are of little interest. The main interest is in what model among those that have a scientific interpretation fits the best.

```
> aout.smaller <- aster(resp ~ varb +
+     fit : (nsloc + ewloc + pop),
+     pred, fam, varb, id, root, data = redata)
> aout.bigger <- aster(resp ~ varb +
+     layer : (nsloc + ewloc + pop),
+     pred, fam, varb, id, root, data = redata)
```

Example Data Analysis (cont.)

```
> anova(aout.smaller, aout, aout.bigger)

Analysis of Deviance Table

Model 1: resp ~ varb + fit:(nsloc + ewloc + pop)
Model 2: resp ~ varb + layer:(nsloc + ewloc) + fit:pop
Model 3: resp ~ varb + layer:(nsloc + ewloc + pop)
  Model Df Model Dev Df Deviance P(>|Chi|)
1       17   -2746.7
2       21   -2712.5  4   34.203 6.772e-07 ***
3       33   -2674.7 12   37.838 0.0001632 ***
---
Signif. codes:
0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
```

Example Data Analysis (cont.)

Despite the largest model fitting the best, we choose the middle model because that one tells us something about fitness directly that the other one does not.

More on this later.

Predicted Values

Now we come to a really hard subject for applied work. Hypothesis tests using the R function `anova` are fairly straightforward. Confidence intervals using the R function `predict` are anything but straightforward (crooked as a dog's hind leg).

Part of this is programmer brain damage (PBD), which is a technical term (an entry in the *Hacker's Dictionary*). The `predict` function has some aspects of its user interface that are clumsy and hard to use, even for the author of the function. Unfortunately, they cannot be fixed without breaking a lot of working examples (which would be much worse than just living with these issues). The R package `aster2` fixes these issues (and does lots more) but is still incomplete.

Predicted Values (cont.)

But the other part of what makes confidence intervals is just the inherent complexity of aster models.

Whatever you personally are trying to do with aster models is a very special case of what aster models can do. As we shall see, they can do many things that look radically different and have no obvious connection with each other. (They don't have any obvious similarities in their data or scientific interpretations of their data. The only connection is aster model theory applies to both.)

Among other issues, aster models have six (!) different parameterizations, *all of which can be of scientific interest in some application*, not necessarily in your application, not necessarily all in any one application.

A Technical Quibble

The R generic function named `predict` does not do prediction except for linear models. What it does do is parameter estimation and confidence intervals for parameters.

So it is misnamed. But that doesn't have anything to do with aster models. `predict` is misnamed even when applied to generalized linear models (GLM).

A referee for the first aster paper complained about this, but we replied that that is just the way R is. The name made some sense when the function was introduced into S in 1988 (before R even existed – S was proprietary software from AT&T that defined the statistical computing language that R is a free software implementation of) because it was mostly just for linear models (although for generalized linear models too, so it was a misnomer even then, but not such an obvious one).

Predicted Values (cont.)

```
> pout <- predict(aout)
> class(pout)

[1] "numeric"

> length(pout)

[1] 5130

> nrow(redata)

[1] 5130
```

Predicted Values (cont.)

`predict.aster` and `predict.aster.formula` have many complicated options. When invoked with no optional arguments (as just shown), it produces a numeric vector of the same length as the response vector.

The result of `predict(aout)` is the maximum likelihood estimate (MLE) of the *saturated model mean value parameter vector* μ.

If y denotes the response vector, then

$$E(y) = \mu$$

meaning

$$E(y_i) = \mu_i$$

(the components of μ are the unconditional expectations of the corresponding components of y).

Predicted Values (cont.)

As everywhere else in statistics we distinguish parameters like μ from their estimates $\hat{\mu}$. We say μ is the unknown true parameter (vector) value that determined the distribution of the data, and $\hat{\mu}$ is only an estimator of μ.

If we want to say how bad or good our estimators are, then we need confidence intervals (or perhaps just standard errors).

```
> pout <- predict(aout, se.fit = TRUE)
> class(pout)

[1] "list"

> sapply(pout, class)

      fit    se.fit  gradient
"numeric" "numeric"  "matrix"
```

Predicted Values (cont.)

The component `fit` gives the estimators (the same vector that was returned when `predict` was invoked with no optional arguments). The component `se.fit` gives the corresponding standard errors.

These are asymptotic (large sample size, approximate) estimated standard deviations of the components of $\hat{\mu}$ derived using the "usual" theory of maximum likelihood estimation (more on that later).

Predicted Values (cont.)

```
> low <- pout$fit - qnorm(0.975) * pout$se.fit
> hig <- pout$fit + qnorm(0.975) * pout$se.fit
> length(hig)
```

```
[1] 5130
```

gives us vector containing confidence bounds for approximate 95% confidence intervals (*not corrected for simultaneous coverage!*) for each of the components of the response vector.

These are of no scientific interest whatsoever.

Predicted Values (cont.)

The question of scientific interest addressed by confidence intervals in the first aster paper was about (best surrogate of) fitness of a *typical* individual in each population. Thus we only want

```
> nlevels(redata$pop)

[1] 7
```

confidence intervals, one for each population.

What do we mean by "typical" individuals? Those that are directly comparable. Those that the same in all respects except for population.

In particular, they should be planted at exactly the same place (have the same values of `nsloc` and `ewloc`). Clearly, real individuals are not comparable in this way. (Two different plants cannot have the same location.)

Predicted Values (cont.)

Thus we have to *make up covariate data* for *hypothetical* individuals that are comparable like this and get estimated mean values for them.

```
> fred <- data.frame(nsloc = 0, ewloc = 0,
+     pop = levels(redata$pop), root = 1,
+     ld02 = 1, ld03 = 1, ld04 = 1,
+     fl02 = 1, fl03 = 1, fl04 = 1,
+     hdct02 = 1, hdct03 = 1, hdct04 = 1)
> fred
```

```
  nsloc ewloc     pop root ld02 ld03 ld04 fl02 fl03
1     0     0      AA    1    1    1    1    1    1
2     0     0  Eriley    1    1    1    1    1    1
3     0     0      Lf    1    1    1    1    1    1
4     0     0 Nessman    1    1    1    1    1    1
5     0     0    NWLF    1    1    1    1    1    1
6     0     0     SPP    1    1    1    1    1    1
```

Predicted Values (cont.)

Seems to work. The components of the response vector are ignored in prediction so we can give them arbitrary values. Somewhat annoyingly, they have to be possible values because `predict.aster.formula` will check.

```
> renewdata <- reshape(fred, varying = list(vars),
+     direction = "long", timevar = "varb",
+     times = as.factor(vars), v.names = "resp")
> layer <- gsub("[0-9]", "", as.character(renewdata$varb))
> renewdata <- data.frame(renewdata, layer = layer)
> fit <- as.numeric(layer == "hdct")
> renewdata <- data.frame(renewdata, fit = fit)
```

We did exactly the same things we did to make `redata` in making `renewdata` changing what had to be changed (*mutatis mutandis* as the economists say).

Predicted Values (cont.)

Now we have predictions for these guys

```
> names(renewdata)

[1] "nsloc" "ewloc" "pop"   "root"  "varb"  "resp"
[7] "id"    "layer" "fit"

> pout <- predict(aout, newdata = renewdata, varvar = varb,
+     idvar = id, root = root, se.fit = TRUE)
> sapply(pout, class)

      fit    se.fit  gradient    modmat
"numeric" "numeric"  "matrix"   "array"

> sapply(pout, length)

     fit   se.fit gradient   modmat
      63       63     1323     1323
```

Predicted Values (cont.)

Why do we need the arguments `varvar`, `idvar`, and `root` when we didn't before? Dunno. More PBD. But `help(predict.aster)` says we need them (especially look at the examples, which is always good advice).

So now we can make 63 not corrected for simultaneous coverage confidence intervals, one for each of the 9 nodes of the graph for each of these 7 individuals (one per population).

These too are of no scientific interest whatsoever. But we are getting closer.

Predicted Values (cont.)

What is of scientific interest is confidence intervals for Darwinian fitness for these 7 individuals.

Fitness (best surrogate of) in these data is the lifetime headcount which is

$$\texttt{hdct02} + \texttt{hdct03} + \texttt{hdct04}$$

where the variable names here are meant to indicate the actual variables.

Wait! What?

$$\texttt{hdct02} + \texttt{hdct03} + \texttt{hdct04}$$

is fitness? What about the other components of fitness? Don't they contribute too?

Yes, they do. But their effect is already counted in the head count. You can't have nonzero head count if you are dead or if you had no flowers, so that is already accounted for.

When we say this, what we mean is that the **unconditional expectation** of head count incorporates the effect of these earlier components of fitness.

Predicted Values (cont.)

Getting the predicted values is no problem if we know the order the nodes of the graph are arranged in, which is shown by

```
> renewdata$id

 [1] 1 2 3 4 5 6 7 1 2 3 4 5 6 7 1 2 3 4 5 6 7 1 2 3 4
[26] 5 6 7 1 2 3 4 5 6 7 1 2 3 4 5 6 7 1 2 3 4 5 6 7 1
[51] 2 3 4 5 6 7 1 2 3 4 5 6 7

> as.character(renewdata$varb)

 [1] "ld02"   "ld02"   "ld02"   "ld02"   "ld02"
 [6] "ld02"   "ld02"   "ld03"   "ld03"   "ld03"
[11] "ld03"   "ld03"   "ld03"   "ld03"   "ld04"
[16] "ld04"   "ld04"   "ld04"   "ld04"   "ld04"
[21] "ld04"   "fl02"   "fl02"   "fl02"   "fl02"
[26] "fl02"   "fl02"   "fl02"   "fl03"   "fl03"
[31] "fl03"   "fl03"   "fl03"   "fl03"   "fl03"
```

Predicted Values (cont.)

We see it runs through all individuals for each node before going on to the next node. So

```
> nnode <- length(vars)
> sally <- matrix(pout$fit, ncol = nnode)
> dim(sally)
```

```
[1] 7 9
```

```
> rownames(sally) <- unique(as.character(renewdata$pop))
> colnames(sally) <- unique(as.character(renewdata$varb))
```

stuffs the parameter estimates into a matrix with individuals along rows and nodes along columns.

Predicted Values (cont.)

```
> round(sally, 4)

           ld02   ld03   ld04   fl02   fl03   fl04
AA       0.7834 0.7521 0.7285 0.3229 0.2560 0.4561
Eriley   0.6954 0.6565 0.6299 0.2334 0.1774 0.3237
Lf       0.7029 0.6646 0.6382 0.2404 0.1834 0.3342
Nessman  0.6377 0.5946 0.5669 0.1824 0.1348 0.2469
NWLF     0.7289 0.6926 0.6670 0.2655 0.2051 0.3716
SPP      0.7937 0.7634 0.7402 0.3346 0.2666 0.4729
Stevens  0.7187 0.6816 0.6557 0.2555 0.1964 0.3567
         hdct02 hdct03 hdct04
AA       0.6215 0.4990 1.2555
Eriley   0.4085 0.3140 0.7796
Lf       0.4242 0.3273 0.8144
Nessman  0.2993 0.2233 0.5418
NWLF     0.4817 0.3764 0.9422
SPP      0.6514 0.5257 1.3224
```

Predicted Values (cont.)

```
> herman <- sally[ , grepl("hdct", colnames(sally))]
> herman
```

```
           hdct02    hdct03    hdct04
AA       0.6215239 0.4990070 1.2554533
Eriley   0.4084934 0.3139651 0.7796097
Lf       0.4242352 0.3272954 0.8144317
Nessman  0.2993480 0.2233300 0.5418002
NWLF     0.4816654 0.3764236 0.9421609
SPP      0.6513874 0.5256736 1.3224073
Stevens  0.4584965 0.3565129 0.8905197
```

```
> rowSums(herman)
```

```
      AA   Eriley        Lf  Nessman     NWLF      SPP
2.375984 1.502068 1.565962 1.064478 1.800250 2.499468
 Stevens
1.705529
```

Predicted Values (cont.)

These are the desired estimates of expected fitness, but they don't come with standard errors because there is no simple way to get the standard errors for sums from the standard errors for the summands (when the summands are not independent, which is the case here).

So we have to proceed indirectly. We have to tell `predict.aster.formula` what functions of mean values we want and let it figure out the standard errors (which it can do).

It only figures out for *linear functions*. We can handle non-linear functions using the delta method "by hand" (using R as a calculator but doing derivatives ourselves), but that is much more complicated. Since addition is a linear operation, we do not need that complication for this example.

Predicted Values (cont.)

If $\hat{\mu}$ is the result of `predict.aster.formula` without the optional argument `amat`, then when the optional argument `amat` is given it does parameter estimates with standard errors for a new parameter

$$\hat{\zeta} = A^T \hat{\mu},$$

where A is a known matrix (the `amat` argument).

Since we want 7 confidence intervals A^T has 7 rows, and since μ is length 63, A^T has 63 columns. Thus A is a 63×7 matrix.

Fairly simple, except now comes some serious PBD.

Predicted Values (cont.)

Quoted from `help(predict.aster)`

> *For `predict.aster`, a three-dimensional array with `dim(amat)[1:2] == dim(modmat)[1:2]`.*
>
> *For `predict.aster.formula`, a three-dimensional array of the same dimensions as required for `predict.aster` (even though `modmat` is not provided). First dimension is number of individuals in `newdata`, if provided, otherwise number of individuals in `object$data`. Second dimension is number of variables (`length(object$pred)`).*

Also clear as mud.

Predicted Values (cont.)

So here is another description. The argument `amat` is a three dimensional array.

The first dimension is the number of individuals in `newdata` (if provided) and otherwise in the `data` argument in the call to `aster` produced the object provided as the first argument to `predict.aster.formula`.

The second dimension is the number of nodes in the graph.

The third dimension is the number parameters we want point estimates and standard errors for.

Predicted Values (cont.)

Let a_{ijk} denote the elements of this array, and let μ_{ij} denote the elements of the result of calling `predict.aster.formula` without the `amat` argument, these elements being stuffed into a matrix columnwise as we showed back on slide 83. Then we are trying to estimate the parameter vector having components

$$\zeta_k = \sum_{i=1}^{n_{\text{ind}}} \sum_{j=1}^{n_{\text{node}}} a_{ijk} \mu_{ij}$$

Predicted Values (cont.)

```
> npop <- nrow(fred)
> nnode <- length(vars)
> amat <- array(0, c(npop, nnode, npop))
> dim(amat)
```

```
[1] 7 9 7
```

We want only the means for the k-th individual to contribute to ζ_k. And we want to add only the headcount entries.

```
> foo <- grepl("hdct", vars)
> for (k in 1:npop)
+     amat[k, foo, k] <- 1
```

This three-way array is too big to print on a slide. We'll just try it out.

Predicted Values (cont.)

```
> pout.amat <- predict(aout, newdata = renewdata, varvar =
+     idvar = id, root = root, se.fit = TRUE, amat = amat)
> pout.amat$fit

[1] 2.375984 1.502068 1.565962 1.064478 1.800250
[6] 2.499468 1.705529

> rowSums(herman)

      AA   Eriley        Lf  Nessman     NWLF      SPP
2.375984 1.502068 1.565962 1.064478 1.800250 2.499468
 Stevens
1.705529
```

Hooray! They're the same!

Predicted Values (cont.)

```
> foo <- cbind(pout.amat$fit, pout.amat$se.fit)
> rownames(foo) <- as.character(fred$pop)
> colnames(foo) <- c("estimates", "std. err.")
> round(foo, 3)
```

```
         estimates std. err.
AA           2.376     0.446
Eriley       1.502     0.196
Lf           1.566     0.249
Nessman      1.064     0.309
NWLF         1.800     0.182
SPP          2.499     0.289
Stevens      1.706     0.222
```

Predicted Values (cont.)

```
> options(show.signif.stars = FALSE)
> anova(aout.smaller, aout, aout.bigger)


Analysis of Deviance Table

Model 1: resp ~ varb + fit:(nsloc + ewloc + pop)
Model 2: resp ~ varb + layer:(nsloc + ewloc) + fit:pop
Model 3: resp ~ varb + layer:(nsloc + ewloc + pop)
  Model Df Model Dev Df Deviance P(>|Chi|)
1       17   -2746.7
2       21   -2712.5  4   34.203 6.772e-07
3       33   -2674.7 12   37.838 0.0001632
```

The only thing that remains to be done is to keep a "more later" promise. In Geyer, et al. (2007) the scientists decided to pick the middle of the three models. How can that be justified?

Predicted Values (cont.)

The justification is that the middle model fits *the quantities of scientific interest* as well as the big model, those quantities being the values of expected fitness for the different pop categories.

```
> levels(redata$pop)
```

```
[1] "AA"      "Eriley"  "Lf"      "Nessman" "NWLF"
[6] "SPP"     "Stevens"
```

Figure 2 in Geyer, et al. (2007) shows that confidence intervals for these quantities of interest differ hardly at all when made with the middle model and when made with the big model. This issue is addressed again in the lecture slides for the special topics course

`http://www.stat.umn.edu/geyer/8931aster/slides/s4.pdf`

slides 74 to the end of deck 4.