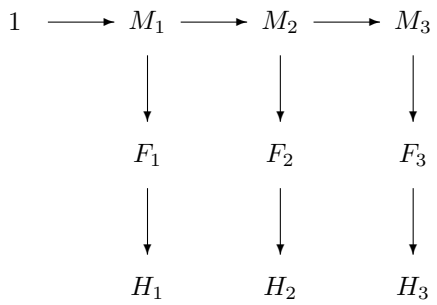
Aster model notes

Daniel J. Eck

Introduction to aster models

Aster models are a class of statistical models designed for life history analysis of plants and animals, that allow joint analysis of data on survival and reproduction over multiple years, allow for variables having different probability distributions, and correctly account for the dependence of variables on earlier variables. These models generalize both generalized linear models and survival analysis. The joint distribution is factorized as a product of conditional distributions, each an exponential family with the conditioning variable being the sample size of the conditional distribution. The model may be heterogeneous, each conditional distribution being from a different exponential family.

We will illustrate the utility of aster models with an analysis of data taken from an experimental study of *Echinacea angustifolia* (aster plants) sampled from remnant prairie populations in western Minnesota. For each individual planted, at each census, we record whether or not it is alive, whether or not it flowers, and its number of flower heads. These data are complicated, especially when recorded for several years, but simple conditional models may suffice. We consider mortality status, dead or alive, to be Bernoulli given the preceding mortality status. Similarly, flowering status given mortality status is also Bernoulli. Given flowering, the number of flower heads may have a zero-truncated Poisson distribution. Figure 1 shows the graphical model for a single individual. Arrows go from parent nodes to child nodes. Nodes are labeled by their associated variables. The only root node is associated with the constant variable 1. M_j is the mortality status in year $2001 + j$. F_j is the flowering status in year $2001 + j$. H_j is the flower head count in year $2001 + j$.The M_j and F_j are Bernoulli conditional on their parent variables being one, and zero otherwise. The H_j are zero-truncated Poisson conditional on their parent variables being one, and zero otherwise.



Aster models are simple graphical model in which the joint density is a product of conditionals as in equation (1) below. Variables in an aster model are denoted by X_j, where j runs over the nodes of a graph. These graphical models follow six assumptions which are:

- **A1** The graph of arrows is acyclic.
- **A2** In the graph of lines every connected component is a complete graph, which is called a dependence group.
- **A3** Every node in a dependence group with more than one node has the same predecessor (there is an arrow from the predecessor to each node in the group). Every dependence group consisting of exactly one node has at most one predecessor.

- **A4** The joint distribution is the product of conditional distributions, one conditional distribution for each dependence group.
- **A5** Predecessor is sample size, meaning each conditional distribution is the distribution of the sum of N independent and identically distributed random vectors, where N is the value of the predecessor, the sum of zero terms being zero.
- **A6** The conditional distributions are exponential families having the components of the response vector for the dependence group as their canonical statistics.

Assumptions A5 and A6 mean for an arrow $X_k \to X_j$ that X_j is the sum of independent and identically distributed random variables from the exponential family for the arrow and there are X_k terms in the sum (the sum of zero terms is zero). These assumptions imply that the joint distribution of the aster model is an exponential family. Three of these assumptions have a clear biological meaning as well. Assumptions A1 through A3 restricts an individual from revisiting life stages that have come to pass. Assumption A5 implies that dead individuals remain dead and have no offspring though the course of the study.

Theory

Factorization

Let F and J respectively denote root and nonroot nodes. Aster models have very special chain graph structure determined by a partition $\mathcal{G}$ of J and a function $p : G \to J \cup F$. For each $G \in \mathcal{G}$ there is an arrow from $p(G)$ to each element of G and a line between each pair of elements of G. For any set S, let X_S denote the vector whose components are $X_j, j \in S$. The graph determines a factorization

$$f(X_J|X_F) = \prod_{G\in\mathcal{G}} f(X_G|X_{p(G)}). \tag{1}$$

We can extend the theory to allow for the elements of X_G to be conditionally dependent given $X_{p(G)}$. Nodes exhibiting this type of dependence are said to constitute a dependency group. An example of this is when an organism may progress down different unique trajectories coded via a multinomial distribution.

Conditional and unconditional distributions

We take each of the conditional distributions in (1) to be an exponential family having canonical statistic X_G that is the sum of $X_{p(G)}$ independent and identically distributed random vectors, possibly a different such family for each G. Conditionally, $X_{p(G)} = 0$ implies that $X_G = 0_G$ almost surely. The log likelihood for the whole family has the form

$$\sum_{G\in\mathcal{G}} \left\{ \sum_{j\in G} X_j\theta_j - X_{p(G)}c_G(\theta_G) \right\} = \sum_{j\in J} X_j\theta_j - \sum_{G\in\mathcal{G}} X_{p(G)}c_G(\theta_G), \tag{2}$$

where θ_G is the canonical parameter vector for the Gth conditional family, having components θ_j, $j \in G$, and c_G is the cumulant function for that family that satisfies

$$\begin{aligned} E_{\theta_G}(X_G|X_{p(G)}) &= X_{p(G)}\nabla c_G(\theta_G) \\ \mathrm{Var}_{\theta_G}(X_G|X_{p(G)}) &= X_{p(G)}\nabla^2 c_G(\theta_G). \end{aligned}$$

Collecting terms with the same X_j in (2), we obtain a convenient form for the joint density

$$\sum_{j\in J} X_j \left\{ \theta_j - \sum_{G\in p^{-1}(\{j\})} c_G(\theta_G) \right\} - \sum_{G\in p^{-1}(F)} X_{p(G)}c_G(\theta_G), \tag{3}$$

and we can take

$$\phi_j = \theta_j - \sum_{G \in p^{-1}(\{j\})} c_G(\theta_G), \qquad j \in J,$$
$$c(\phi) = \sum_{G \in p^{-1}(F)} X_{p(G)} c_G(\theta_G),$$

where the $X_{p(G)}$ terms that form $c(\phi)$ are at the root nodes, and hence are nonrandom, so that $c(\phi)$ is a deterministic cumulant function. Thus, (3) can be written as

$$l(\phi) = \langle X, \phi \rangle - c(\phi), \tag{4}$$

and we are back where we started! The aster model in the ϕ parameterization has the same log likelihood as the generic exponential families that we have studied throughout this course. We refer to ϕ as a saturated aster model unconditional canonical parameter vector. Note that the system of equations

$$\phi_j = \theta_j - \sum_{G \in p^{-1}(\{j\})} c_G(\theta_G), \qquad j \in J,$$

can be solved for the θ_j in terms of the ϕ_j in one pass through the equations in any order that finds θ_j for children before parents. Thus switching from θ to ϕ determines an invertible change of parameters. This change of parameterizations is refered to as the aster transform. An example of this is given in Section 2.5 of this technical report.

Aster submodels and parameterizations

The aster model with log likelihood (4) is not useful because it is saturated, there is a parameter for every individual and every node in the aster graph. We can consider affine submodels of the form

$$\phi = a + M\beta$$

where a is a known offset vector and M is a known model matrix. In these notes we will ignore the offset vector unless otherwise specified and only consider liner submodels of the form

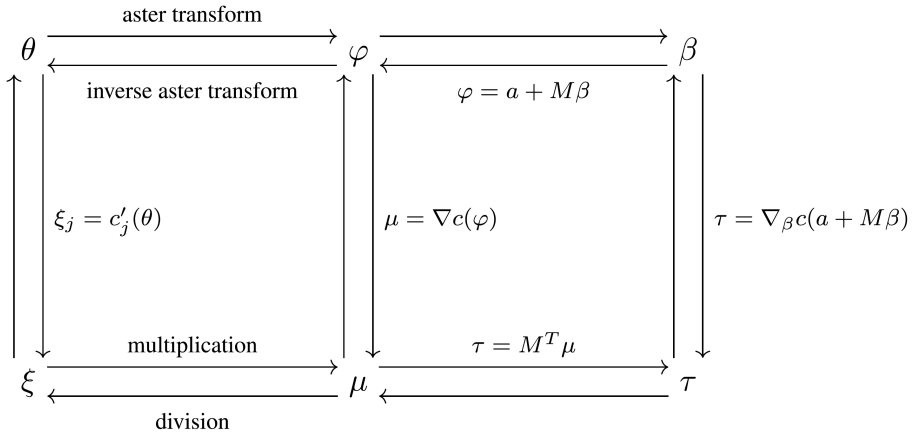
$$\phi = M\beta.$$

This results in a new exponential family with canonical statistic $Y = M^T X$ and unconditional submodel canonical parameter vector β. We can write the log likelihood for this submodel as

$$l(\beta) = \langle Y, \beta \rangle - c_{\text{sub}}(\beta)$$

where $c_{\text{sub}} = c(M\beta)$. The exponential family form allows us to conveniently obtain the maximum likelihood estimator for our canonical parameter vector β using conventional software.

Like the exponential families that we motivated throughout this course there are a plethora of aster model parameterizations. We already saw the unconditional saturated aster model canonical parameter vector ϕ and corresponding submodel parameter vector β. We also motivated the conditional aster model canonical parameter vector θ, and we noted that ϕ and θ are invertible parameterizations. We also have the conditional saturated aster model mean-value parameter vector ξ, the unconditional saturated aster model mean-value parameter vector μ, and the unconditional aster sub model mean-value parameter vector τ. A depiction of these parameterizations and their relationships between each other is given below. Primary interest will be given to unconditional aster model parameterizations.



Asymptotic properties and inference

Since we have a log likelihood in the form of a canonical exponential family submodel

$$l(\beta) = \langle Y, \beta \rangle - c_{\text{sub}}(\beta),$$

we can obtain a maximum likelihood estimator $\hat{\beta}$ such that

$$\sqrt{n}(\hat{\beta} - \beta) \to N(0, \Sigma^{-1})$$

where Σ is the Fisher information matrix corresponding to $l(\beta)$. The `aster` package [Geyer, 2019] provides estimates of $\hat{\beta}$ and $\widehat{\Sigma}$.

In scientific applications μ, or a function of μ, will be the parameter of interest. In many aster analyses one seeks to estimate expected Darwinian fitness (total number of offspring produced by individuals) as a function of covariates. One may estimate a proxy for expected Darwinian fitness when one does not have this information recorded. The sum of flower head counts $\sum_j H_j$ can be taken as a proxy in the motivating *E. angustifolia* example. In any event, suppose the parameter of interest is given by $g(\beta)$. Then this parameter has an estimator with the following asymptotic distribution

$$\sqrt{n}(g(\hat{\beta}) - g(\beta)) \to N\left(0, \nabla g(\beta)\Sigma^{-1}\{\nabla g(\beta)\}^T\right).$$

Echinacea example

The Echinacea Data for are found in the R dataset `echinacea` in the R package `aster` [Geyer, 2019]. We see a part of this data below.

Initial data processing

```
library(aster)
```

```
## Loading required package: trust
```

```
data(echinacea)
head(echinacea)
```

```
##   hdct02 hdct03 hdct04    pop ewloc nsloc ld02 fl02 ld03 fl03 ld04 fl04
## 1      0      0      0   NWLF    -8   -11    0    0    0    0    0    0
## 2      0      0      0 Eriley    -8   -10    1    0    1    0    1    0
## 3      0      0      0   NWLF    -8    -9    0    0    0    0    0    0
## 4      0      0      0    SPP    -8    -8    0    0    0    0    0    0
## 5      0      0      0    SPP    -8    -7    0    0    0    0    0    0
## 6      0      0      1 Eriley    -8    -6    1    0    1    0    1    1
```

```
names(echinacea)
```

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

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 predecessor of 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. Let's check to see if this makes sense.

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

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

That's right.

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

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.

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

```
##           pop ewloc nsloc varb resp id root
## 1.ld02   NWLF    -8   -11 ld02    0  1    1
## 2.ld02 Eriley    -8   -10 ld02    1  2    1
## 3.ld02   NWLF    -8    -9 ld02    0  3    1
## 4.ld02    SPP    -8    -8 ld02    0  4    1
## 5.ld02    SPP    -8    -7 ld02    0  5    1
## 6.ld02 Eriley    -8    -6 ld02    1  6    1
```

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" "hdct04" "ld02"   "ld03"
## [9] "ld04"
```

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?

We defined `root` 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. The `id` variable is seldom (if ever) used. It tells what individual (what row of the original data frame `echinacea`) a row of reshape came from. The variables `nsloc` (north-south location) and `ewloc` (east-west location) give the position each individual was located in the experimental plot. The variable `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"      "NWLF"    "Nessman" "SPP"     "Stevens"
```

Fitting aster models

We will now discuss fitting aster 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

$$\texttt{y} \sim \texttt{varb + ...}$$

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.

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

$$\texttt{y} \sim \texttt{varb + varb:(...)}$$

where ... is any other part of the formula. 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. That being said, formulas like this would yield too many regression coefficients to estimate well. We can do better! Maybe we do not really need different regression coefficients for each node. Maybe different for each kind of node (whatever that may mean) would be enough.

```
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] "character"
```

Maybe

$$\texttt{y} \sim \texttt{varb + layer:(...)}$$

is good enough? But formulas like this would still yield too many regression coefficients to estimate well. We can do better!

In aster models (and this is really where the explanation gets dumbed down to the point of being wrong) 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?

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

```
## [1] 0 1
```

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

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

Maybe

$$\texttt{y} \sim \texttt{varb + fit:(...)}$$

is good enough? We called the variable we just made up `fit` which is short for Darwinian fitness. The regression coefficients in terms specified by ... have a direct relationship with expected Darwinian fitness (or a surrogate of Darwinian fitness). And that is usually what is wanted in life history analysis. We now fit our first aster model.

```
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 = id,
##     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
## varbld03         1.740051   0.396189   4.392 1.12e-05 ***
## varbld04         4.188577   0.334266  12.531  < 2e-16 ***
## layerfl:nsloc    0.070102   0.014652   4.785 1.71e-06 ***
## layerhdct:nsloc -0.005804   0.005550  -1.046   0.2956
## layerld:nsloc    0.007165   0.005867   1.221   0.2220
## layerfl:ewloc    0.017977   0.014413   1.247   0.2123
## layerhdct:ewloc  0.007606   0.005561   1.368   0.1714
## layerld:ewloc   -0.004787   0.005919  -0.809   0.4186
## fit:popAA        0.129238   0.089129   1.450   0.1471
## fit:popEriley   -0.049561   0.071279  -0.695   0.4869
## fit:popLf       -0.033279   0.079573  -0.418   0.6758
## fit:popNWLF      0.021028   0.063600   0.331   0.7409
## fit:popNessman  -0.186269   0.127787  -1.458   0.1449
## fit:popSPP       0.149179   0.067716   2.203   0.0276 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Original predictor variables dropped (aliased)
##      fit:popStevens
```

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)
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
```

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.

The argument for doing this is because we are interested in modeling fitness, and the distribution of fitness (actually best surrogate of fitness in their data) is not very different between the two models. The distribution of other components of fitness (other than the final one) may differ quite a lot, but that was not the question of scientific interest. So what do these models say about the distribution of fitness?

```
## we will go over this later
pop <- levels(redata$pop)
nind <- length(unique(redata$id))
nnode <- nlevels(redata$varb)
```

```
npop <- length(pop)
amat <- array(0, c(nind, nnode, npop))
amat.ind <- array(as.character(redata$pop),
  c(nind, nnode, npop))
amat.node <- array(as.character(redata$varb),
  c(nind, nnode, npop))
amat.fit <- grepl("hdct", amat.node)
amat.fit <- array(amat.fit,
  c(nind, nnode, npop))
amat.pop <- array(pop, c(npop, nnode, nind))
amat.pop <- aperm(amat.pop)
amat[amat.pop == amat.ind & amat.fit] <- 1
pout <- predict(aout,  varvar = varb, idvar = id,
  root = root, se.fit = TRUE, amat = amat)
pout.bigger <- predict(aout.bigger, varvar = varb,
  idvar = id, root = root, se.fit = TRUE, amat = amat)
```

The first interesting thing about these *predictions* (actually point estimates of parameters with standard errors) is that the point estimates are exactly the same for the two models.

```
pout$fit
```

```
## [1]  81 171 112 286  31 218 167
```

```
pout.bigger$fit
```

```
## [1]  81 171 112 286  31 218 167
```

```
all.equal(pout$fit, pout.bigger$fit)
```

```
## [1] TRUE
```

And why is that? These are submodel canonical statistics (components of $M^T y$). Thus by the observed-equals-expected property of exponential families their MLE are equal to their observed values and hence equal to each other. So that is certainly not a reason to prefer one model to the other. If the estimated means are exactly the same how about estimated asymptotic variances?

The asymptotic variance matrix of these canonical statistics is actually diagonal for each model. The reason is that different populations of origin have different individuals in the sample, and only individuals from one population contribute to estimating one of these canonical statistics. Thus it is enough to look at the asymptotic standard errors (all the covariances are zero).

```
pout$se.fit
```

```
## [1] 13.617532 19.984170 16.267065 25.968492  8.524453 22.227096 19.884556
```

```
pout.bigger$se.fit
```

```
## [1] 14.521691 17.870387 14.513433 27.857509  9.105173 21.589790 21.642168
```

We see that they are not that different.

If we were interested in the effect of population on the different components of fitness, then the P-value 0.00016 does indicate that the model `aout.bigger`, which has different population effects in different *layers* of the graph, does show a statistically significant difference in the way the components of fitness combine to make up fitness in the various population of origin groups. But if we are only interested in overall fitness rather than the separate components, then there is hardly any difference in the two models.

Estimating expected Darwinian fitness

Hypothesis tests using the R function `anova` are fairly straightforward. Confidence intervals using the R function `predict` for estimates of expected Darwinian fitness are anything but straightforward. Part of this is bad design. The `predict` function has some aspects of its user interface that are clumsy and hard to use (Charlie's words, not mine). The R package `aster2` fixes these issues (and does lots more) but is still incomplete. But the other part of what makes confidence intervals is just the inherent complexity of aster models. 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.

The result of `predict(aout)` is the maximum likelihood estimate of the saturated model mean value parameter vector μ. If we want to say how bad or good our estimators are, then weneed confidence intervals (or perhaps just standard errors).

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

The components of `predict(aout)` are

- 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.
- The component `gradient` gives the derivative of the map from regression coefficients to predictions.

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.

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

The above are 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. The question of scientific interest addressed by confidence intervals 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. Thus we have to make up covariate data for hypothetical individuals that are comparable like this and get estimated mean values for them.

```
dat <- 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)
dat
```

```
##   nsloc ewloc     pop root ld02 ld03 ld04 fl02 fl03 fl04 hdct02 hdct03 hdct04
## 1     0     0      AA    1    1    1    1    1    1    1      1      1      1
## 2     0     0  Eriley    1    1    1    1    1    1    1      1      1      1
## 3     0     0      Lf    1    1    1    1    1    1    1      1      1      1
## 4     0     0    NWLF    1    1    1    1    1    1    1      1      1      1
## 5     0     0 Nessman    1    1    1    1    1    1    1      1      1      1
## 6     0     0     SPP    1    1    1    1    1    1    1      1      1      1
## 7     0     0 Stevens    1    1    1    1    1    1    1      1      1      1
```

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(dat, 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)
```

Now we have predictions for these variables

```
names(renewdata)
```

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

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

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

Why do we need the arguments `varvar`, `idvar`, and `root` when we did not before? More bad design (Charlie's words, not mine). 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 hypothetical individuals (one per population). These too are of no scientific interest whatsoever. But we are getting closer.

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

```
hdct02 + hdct03 + hdct04
```

where the variable names here are meant to indicate the actual variables. 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 cannot have nonzero head count if you are dead or if you had no flowers, so that is already accounted for.

We now obtain estimates of μ for each hypothetical individual, different rows for different individuals.

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

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

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

```
##              ld02      ld03      ld04      fl02      fl03      fl04    hdct02
## AA      0.7833884 0.7521016 0.7284738 0.3228949 0.2560121 0.4560906 0.6215239
## Eriley  0.6954310 0.6565099 0.6299422 0.2333911 0.1774180 0.3236674 0.4084934
## Lf      0.7029431 0.6646121 0.6382397 0.2404182 0.1834251 0.3342160 0.4242352
## NWLF    0.7288502 0.6926410 0.6670184 0.2654522 0.2050578 0.3716382 0.4816654
## Nessman 0.6377067 0.5946209 0.5668688 0.1824325 0.1347627 0.2469403 0.2993480
## SPP     0.7936647 0.7633787 0.7401963 0.3345782 0.2665928 0.4729477 0.6513874
## Stevens 0.7186716 0.6816127 0.6556811 0.2554631 0.1963829 0.3567396 0.4584965
##            hdct03    hdct04
## AA      0.4990070 1.2554533
## Eriley  0.3139651 0.7796097
## Lf      0.3272954 0.8144317
```

```
## NWLF    0.3764236 0.9421609
## Nessman 0.2233300 0.5418002
## SPP     0.5256736 1.3224073
## Stevens 0.3565129 0.8905197
```

We now obtain estimated expected Dawinian fitness for typical individuals belonging to each population.

```
preds_hdct <- preds[ , grepl("hdct", colnames(preds))]
rowSums(preds_hdct)
```

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

These are the desired estimates of expected fitness, but they do not 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.

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). The argument `amat` is a three dimensional array. The first dimension is the number of individuals (in `newdata` if provided, and otherwise in the original data). The second dimension is the number of nodes in the graph. The third dimension is the number of parameters we want point estimates and standard errors for.

```
npop <- nrow(dat)
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. And we want to add only the head count entries.

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

We now obtain estimates of expected Darwinian fitness and its standard error using `predict.aster`.

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

## predict.aster
pout.amat$fit
```

```
## [1] 2.375984 1.502068 1.565962 1.800250 1.064478 2.499468 1.705529
```

```
## computation by hand
rowSums(preds_hdct)
```

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

Here are the estimated standard errors corresponding to estimates of expected Darwinian fitness for hypothetical typical individuals belonging to each population.

```r
foo <- cbind(pout.amat$fit, pout.amat$se.fit)
rownames(foo) <- unique(as.character(renewdata$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
## NWLF        1.800     0.182
## Nessman     1.064     0.309
## SPP         2.499     0.289
## Stevens     1.706     0.222
```

Acknowledgments

These notes borrow materials from Geyer et al. [2007], Shaw et al. [2008], Eck [2017], Eck et al. [2020], and Charlie Geyer's course slides

References

Daniel J Eck. Statistical inference in multivariate settings. 2017.

Daniel J Eck, Charles J Geyer, and R Dennis Cook. Combining envelope methodology and aster models for variance reduction in life history analyses. *Journal of Statistical Planning and Inference*, 205:283–292, 2020.

Charles J Geyer. Package 'aster'. 2019.

Charles J Geyer, Stuart Wagenius, and Ruth G Shaw. Aster models for life history analysis. *Biometrika*, 94 (2):415–426, 2007.

Ruth G Shaw, Charles J Geyer, Stuart Wagenius, Helen H Hangelbroek, and Julie R Etterson. Unifying life-history analyses for inference of fitness and population growth. *The American Naturalist*, 172(1): E35–E47, 2008.