STAT 528 - Advanced Regression Analysis II

Aster models

Daniel J. Eck
Department of Statistics
University of Illinois

Learning Objectives Today

- aster model example
- aster analysis

The variables under consideration:

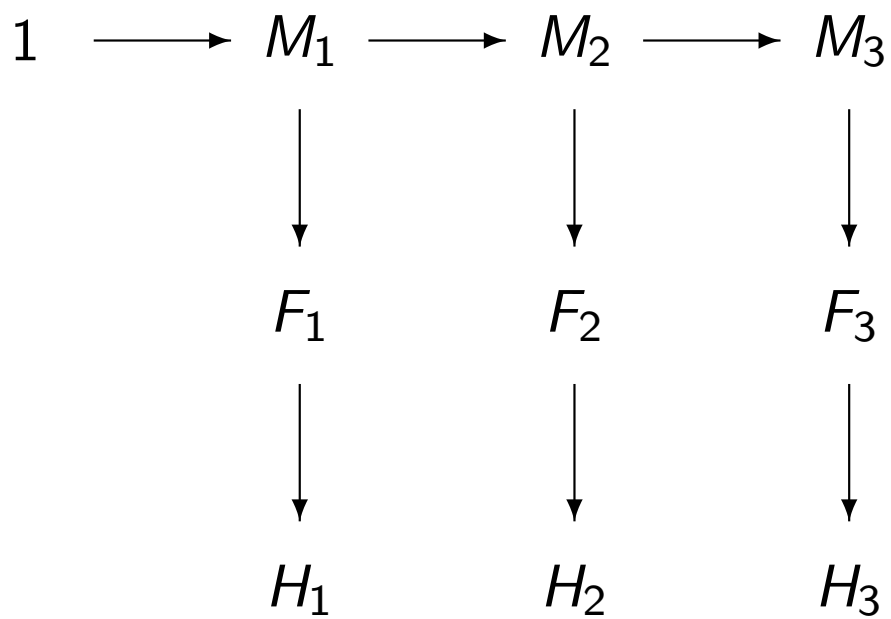
- `nsloc` north-south location of each individual in the experimental plot
- `ewloc` east-west location of each individual in the experimental plot
- `pop` the ancestral population of each individual

Each individual was grown from seed taken from a surviving population in a prairie remnant in western Minnesota near the Echinacea Project field site.

Darwinian fitness (our best surrogate of Darwinian fitness) is total flower head count over the years of data collection.

We are interested in estimated expected Darwinian fitness for the different ancestral populations.

The aster graph for *Echinacea angustifolia* (aster plants)



We load in necessary packages:

```
library(tidyverse)
library(ggplot2)
library(aster)
library(aster2)
```

Initial data processing

Here is a brief look at the data:

```
data("echinacea")
names(echinacea)
```

```
##  [1] "redata"        "repred"        "regroup"       "recode"
##  [5] "families"      "redelta"       "initial"       "response.name"
##  [9] "pred"          "group"         "code"
```

```
head(echinacea$redata)
```

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

```
echinacea$redata %>% filter(id == 1)
```

```
##           pop ewloc nsloc   varb resp id
## 1.ld02   NWLF    -8   -11   ld02    0  1
## 1.ld03   NWLF    -8   -11   ld03    0  1
## 1.ld04   NWLF    -8   -11   ld04    0  1
## 1.fl02   NWLF    -8   -11   fl02    0  1
## 1.fl03   NWLF    -8   -11   fl03    0  1
## 1.fl04   NWLF    -8   -11   fl04    0  1
## 1.hdct02 NWLF    -8   -11 hdct02    0  1
## 1.hdct03 NWLF    -8   -11 hdct03    0  1
## 1.hdct04 NWLF    -8   -11 hdct04    0  1
```

```
echinacea$redata %>% filter(id == 6)
```

```
##             pop ewloc nsloc   varb resp id
## 6.ld02   Eriley    -8    -6   ld02    1  6
## 6.ld03   Eriley    -8    -6   ld03    1  6
## 6.ld04   Eriley    -8    -6   ld04    1  6
## 6.fl02   Eriley    -8    -6   fl02    0  6
## 6.fl03   Eriley    -8    -6   fl03    0  6
## 6.fl04   Eriley    -8    -6   fl04    1  6
## 6.hdct02 Eriley    -8    -6 hdct02    0  6
## 6.hdct03 Eriley    -8    -6 hdct03    0  6
## 6.hdct04 Eriley    -8    -6 hdct04    1  6
```

We can see the proportion of individuals that survive each year.

```
## M1
echinacea$redata %>% filter(varb == "ld02") %>% pull(resp) %>% table()
```

```
## .
##   0   1
## 158 412
```

```
## M2
echinacea$redata %>% filter(id %in% (echinacea$redata %>%
                                      filter(varb == "ld02" & resp == 1) %>%
                                      pull(id)) & varb == "ld03") %>%
  pull(resp) %>% table()
```

```
## .
##   0   1
##  20 392
```

```
## M3
echinacea$redata %>% filter(id %in% (echinacea$redata %>%
                                      filter(varb == "ld03" & resp == 1) %>%
                                      pull(id)) & varb == "ld04") %>%
  pull(resp) %>% table()
```

```
## .
##   0   1
##  14 378
```

We can see the proportion of individuals that flower each year.

```
## F1
echinacea$redata %>% filter(id %in% (echinacea$redata %>%
                                      filter(varb == "ld02" & resp == 1) %>%
                                      pull(id)) & varb == "fl02") %>%
  pull(resp) %>% table()
```

```
## .
##   0   1
## 253 159
```

```
## F2
echinacea$redata %>% filter(id %in% (echinacea$redata %>%
                                      filter(varb == "ld03" & resp == 1) %>%
                                      pull(id)) & varb == "fl03") %>%
  pull(resp) %>% table()
```

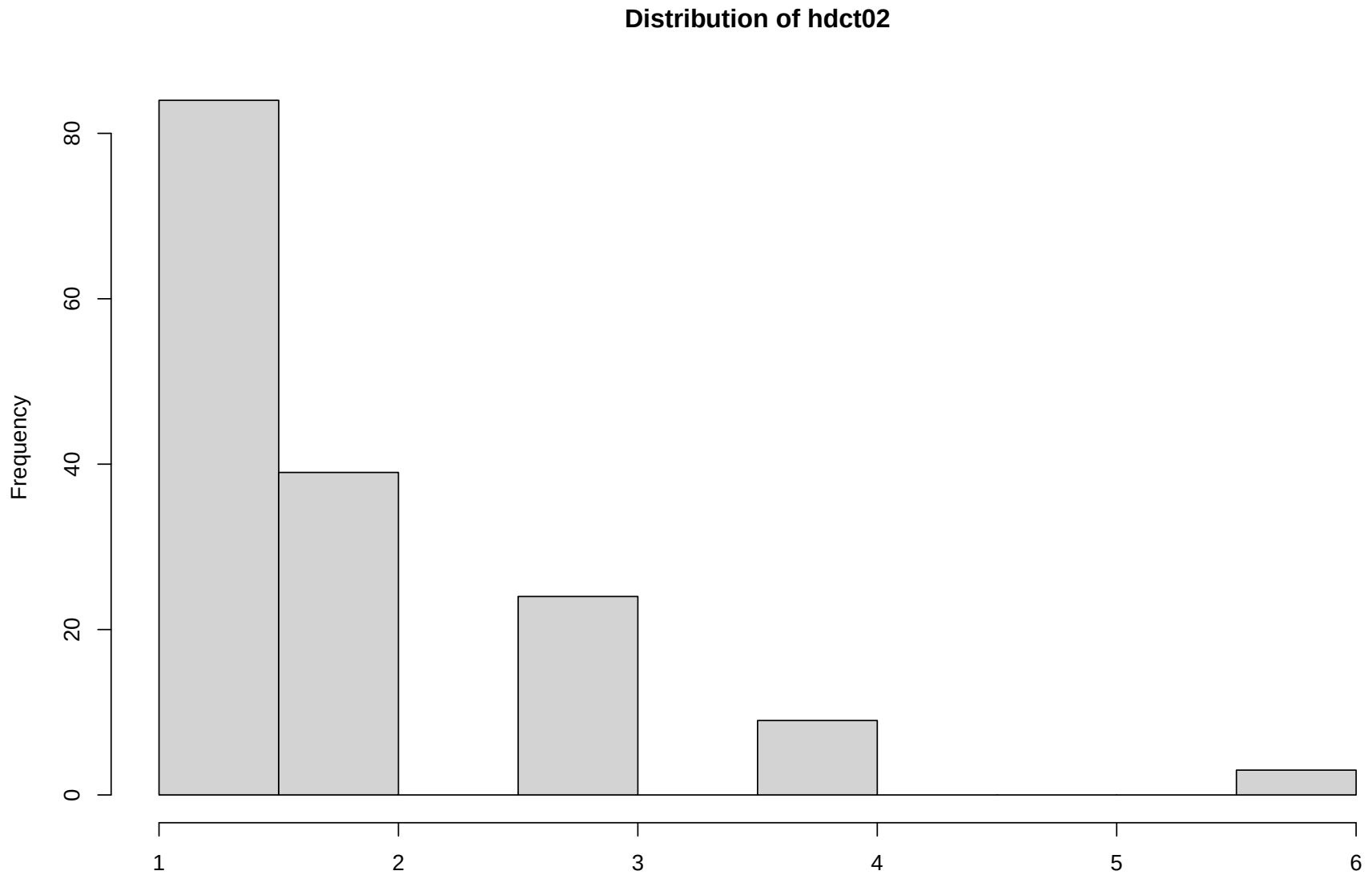
```
## .
##   0   1
## 266 126
```

```
## F3
echinacea$redata %>% filter(id %in% (echinacea$redata %>%
                                      filter(varb == "ld04" & resp == 1) %>%
                                      pull(id)) & varb == "fl04") %>%
  pull(resp) %>% table()
```

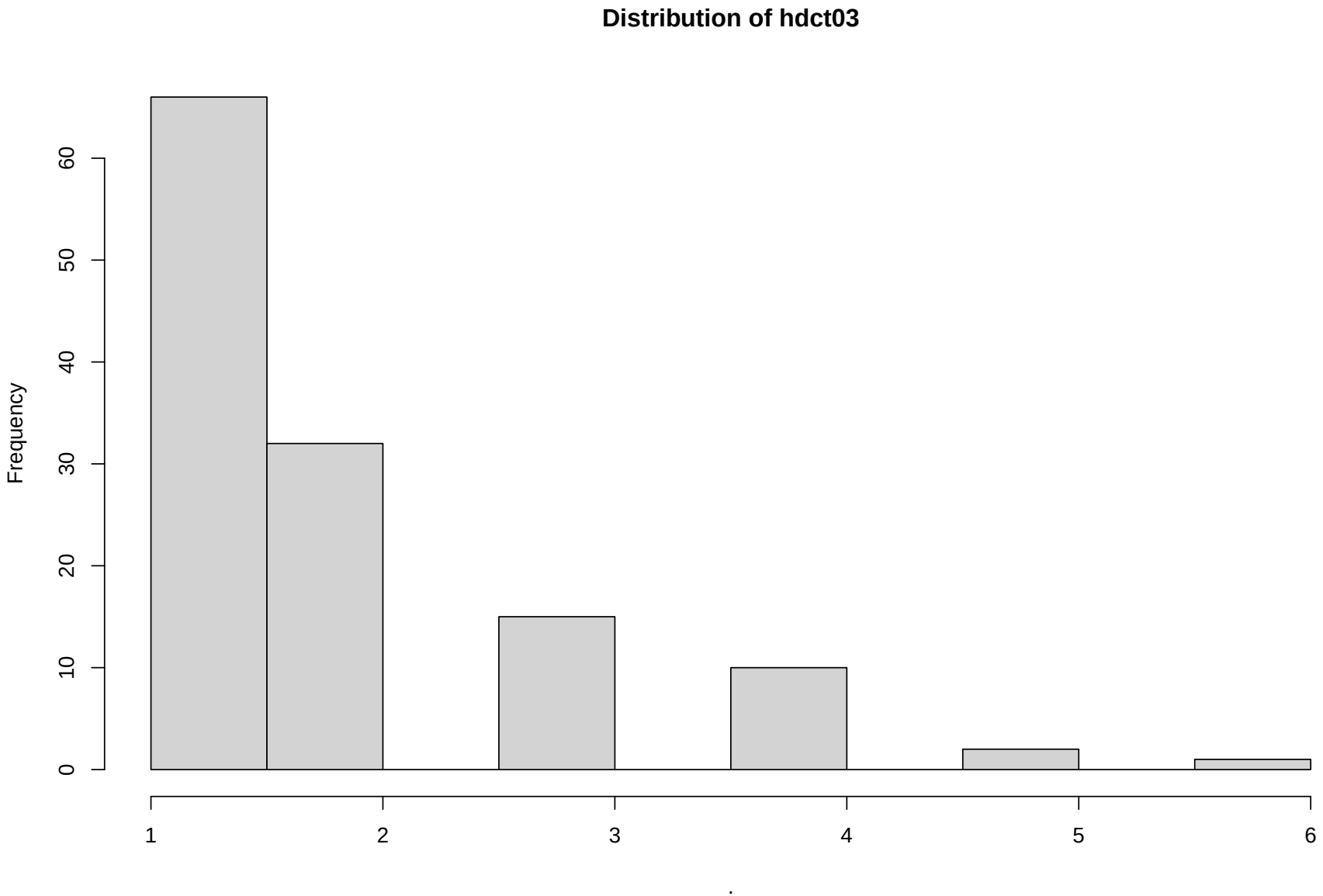
```
## .
##   0   1
## 162 216
```

We can see the distribution of head counts each year.

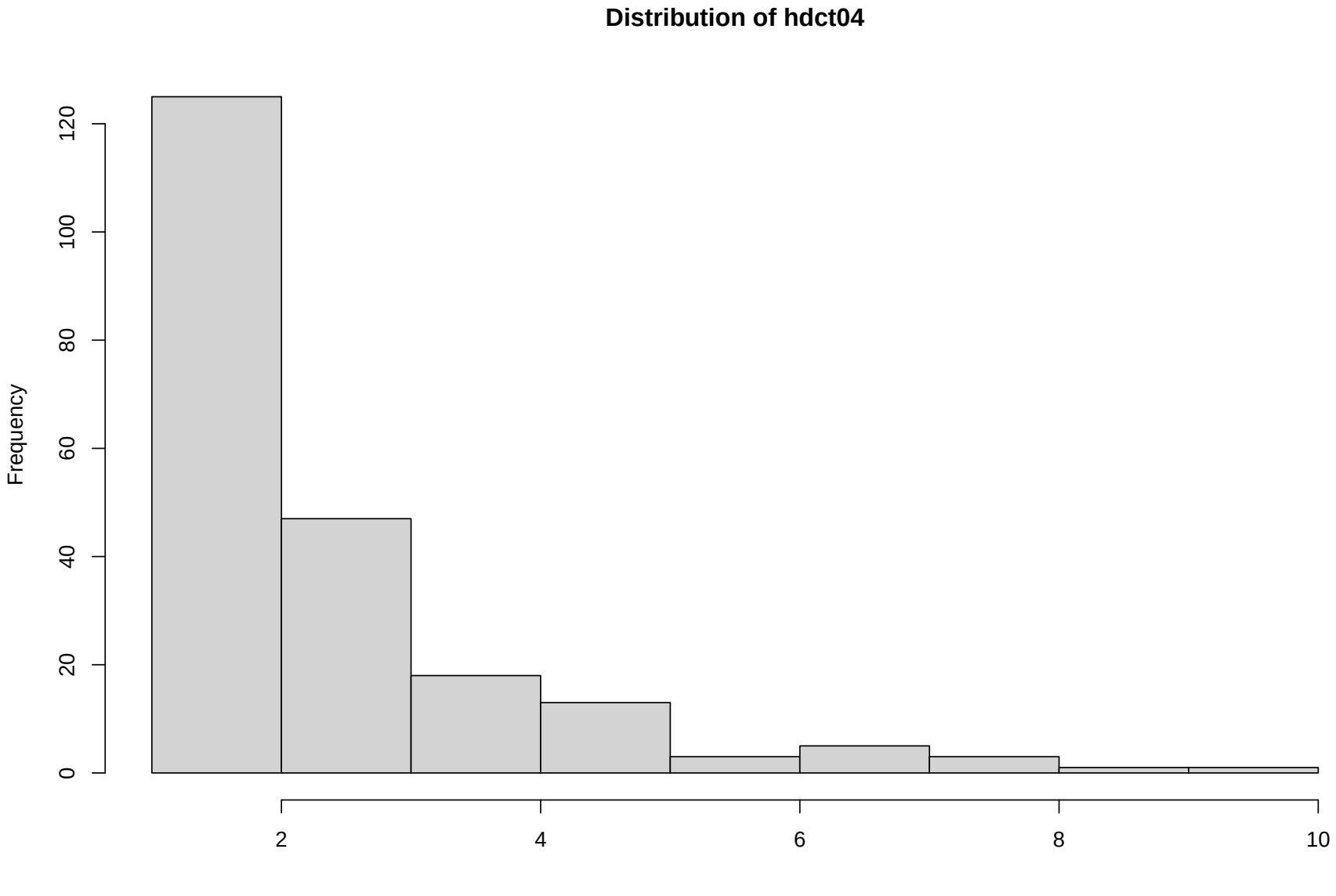
```
echinacea$redata %>% filter(id %in% (echinacea$redata %>%
                                      filter(varb == "fl02" & resp == 1) %>%
                                      pull(id)) & varb == "hdct02") %>%
  pull(resp) %>% hist(., main = "Distribution of hdct02")
```



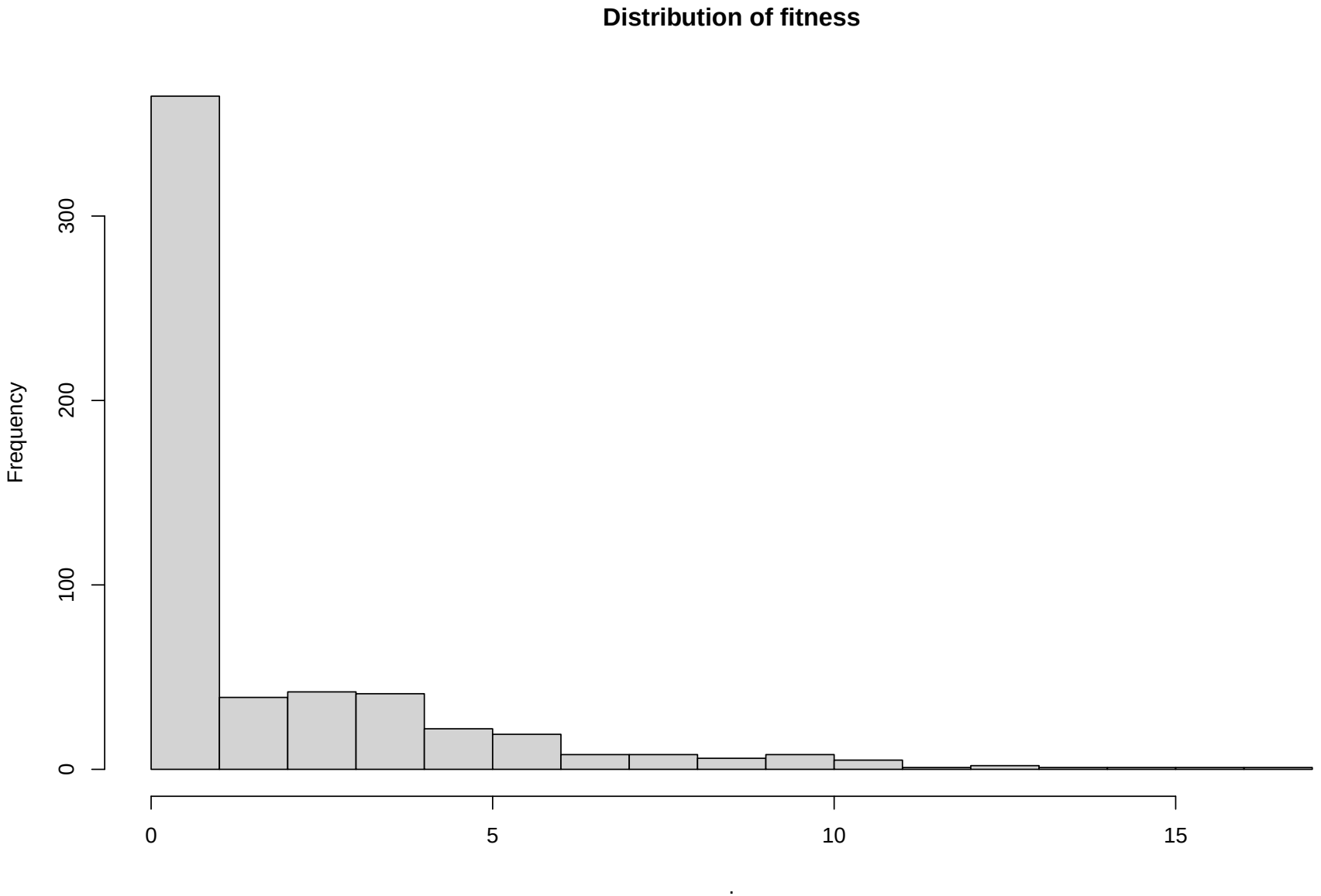
```
echinacea$redata %>% filter(id %in% (echinacea$redata %>%
                                      filter(varb == "fl03" & resp == 1) %>%
                                      pull(id)) & varb == "hdct03") %>%
  pull(resp) %>% hist(., main = "Distribution of hdct03")
```



```
echinacea$redata %>% filter(id %in% (echinacea$redata %>%
                                      filter(varb == "fl04" & resp == 1) %>%
                                      pull(id)) & varb == "hdct04") %>%
  pull(resp) %>% hist(., main = "Distribution of hdct04")
```



```
echinacea$redata %>% group_by(id) %>%
  filter(varb %in% c("hdct02","hdct03","hdct04")) %>%
  summarise(fitness = sum(resp)) %>%
  pull(fitness) %>% hist(., main = "Distribution of fitness", breaks = 20)
```



Aster analysis preliminaries

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.

```
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.

```
## aster example already in long format
redata <- data.frame(echinacea$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] "ld02"   "ld03"   "ld04"   "fl02"   "fl03"   "fl04"   "hdct02" "hdct03"
## [9] "hdct04"
```

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

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

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.

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 terms in the model but rather as specifying separate coefficients for the terms in the model for each node of the graph.

That being said, formulas like this would likely yield too many regression coefficients to estimate well.

Maybe different coefficients for each kind of node (ie mortality or head count) would be good enough.

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

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

Maybe

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

is good enough? But formulas like this would still yield too many regression coefficients to estimate well.

In aster models 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")
redata <- data.frame(redata, fit = fit)
unique(fit)
```

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

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)     -1.079946   0.241164  -4.478 7.53e-06 ***
## varbld03         1.769353   0.529200   3.343 0.000827 ***
## varbld04         4.217879   0.368282  11.453  < 2e-16 ***
## varbfl02         0.029302   0.315703   0.093 0.926050
## varbfl03        -0.319794   0.316120  -1.012 0.311720
## varbfl04        -0.314920   0.295018  -1.067 0.285763
## varbhdct02       1.350716   0.259254   5.210 1.89e-07 ***
## varbhdct03       1.372676   0.262255   5.234 1.66e-07 ***
## varbhdct04       1.880630   0.251000   7.493 6.75e-14 ***
## layerfl:nsloc    0.070102   0.014652   4.785 1.71e-06 ***
## layerhdct:nsloc -0.005804   0.005550  -1.046 0.295638
## layerld:nsloc    0.007165   0.005867   1.221 0.221957
## layerfl:ewloc    0.017977   0.014413   1.247 0.212294
## layerhdct:ewloc  0.007606   0.005561   1.368 0.171381
## layerld:ewloc   -0.004787   0.005919  -0.809 0.418635
## fit:popAA        0.129238   0.089129   1.450 0.147058
## fit:popEriley   -0.049561   0.071279  -0.695 0.486858
## fit:popLf       -0.033279   0.079573  -0.418 0.675789
## fit:popNessman  -0.186269   0.127787  -1.458 0.144936
## fit:popNWLF      0.021028   0.063600   0.331 0.740920
## fit:popSPP       0.149179   0.067716   2.203 0.027593 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
```

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  31 286 218 167
```

```
pout.bigger$fit
```

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

```
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  8.524453 25.968492 22.227096 19.884556
```

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

```
## [1] 14.521691 17.870387 14.513433  9.105173 27.857509 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` fits the data better.

The model `aout.bigger` 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.

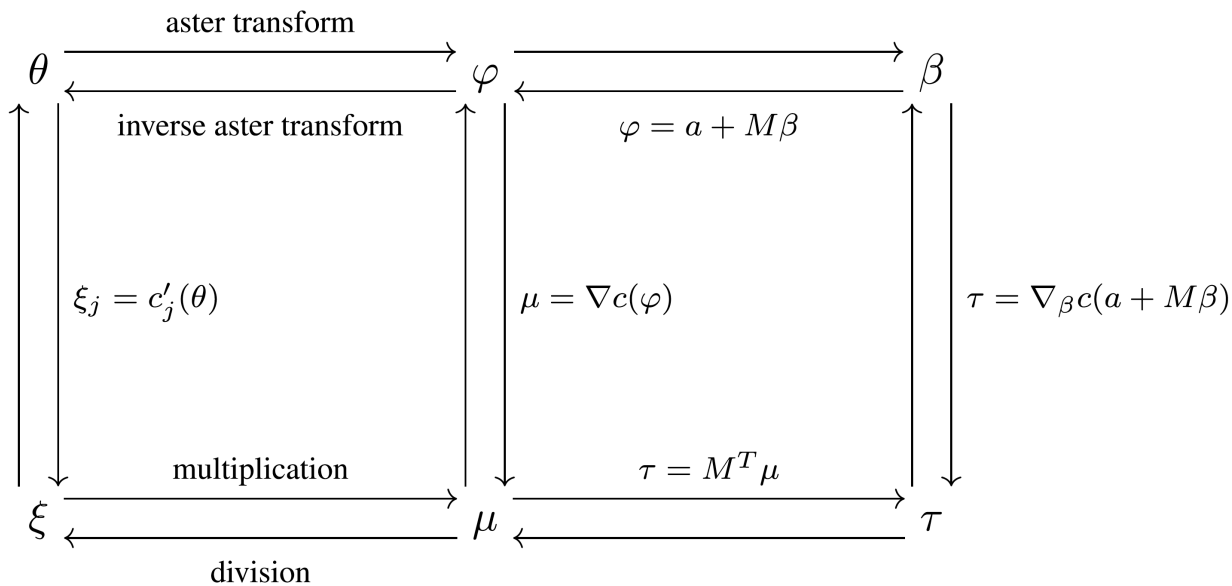
But if we are only interested in overall fitness rather than the separate components, then there is hardly any difference in the two models.

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.

Among other issues, aster models have six different parameterizations, all of which can be of scientific interest in some applications.


aster transform
θ
φ
β
inverse aster transform
$\varphi = a + M\beta$
$\xi_j = c'_j(\theta)$
$\mu = \nabla c(\varphi)$
$\tau = \nabla_\beta c(a + M\beta)$
multiplication
$\tau = M^T\mu$
ξ
μ
τ
division

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 we need 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.

In any event, suppose the parameter of interest is given by $h(\beta)$. Then this parameter has an estimator with the following asymptotic distribution

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

Below are confidence bounds for approximate 95% confidence intervals (not corrected for simultaneous coverage) for each of the components of the response vector.

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

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 Nessman    1    1    1    1    1    1    1      1      1      1
## 5     0     0    NWLF    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.

We now wrangle this new data into a format to be used by `predict.aster`.

```
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)
head(renewdata)
```

```
##        nsloc ewloc     pop root varb resp id layer fit
## 1.ld02     0     0      AA    1 ld02    1  1    ld   0
## 2.ld02     0     0  Eriley    1 ld02    1  2    ld   0
## 3.ld02     0     0      Lf    1 ld02    1  3    ld   0
## 4.ld02     0     0 Nessman    1 ld02    1  4    ld   0
## 5.ld02     0     0    NWLF    1 ld02    1  5    ld   0
## 6.ld02     0     0     SPP    1 ld02    1  6    ld   0
```

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 Geyer'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
```

The effects of other components of fitness is already counted in 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
## Nessman 0.6377067 0.5946209 0.5668688 0.1824325 0.1347627 0.2469403 0.2993480
## NWLF    0.7288502 0.6926410 0.6670184 0.2654522 0.2050578 0.3716382 0.4816654
## 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
## Nessman 0.2233300 0.5418002
## NWLF    0.3764236 0.9421609
## SPP     0.5256736 1.3224073
## Stevens 0.3565129 0.8905197
```

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

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

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

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

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

```
mean_value <- cbind(pout.amat$fit, pout.amat$se.fit)
rownames(mean_value) <- unique(as.character(renewdata$pop))
colnames(mean_value) <- c("estimates", "std. err.")
round(mean_value, 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
```

We can obtain estimates of submodel conditional mean-value parameters (ie mean survival for each ancestral line).

```
pout_cond <- predict(aout, newdata = renewdata, varvar = varb,
  idvar = id, root = root, se.fit = TRUE,
  model.type = "unconditional", parm.type = "mean.value")

nnode <- length(vars)
preds_cond <- matrix(pout_cond$fit, ncol = nnode)
rownames(preds_cond) <- pop
colnames(preds_cond) <- vars
preds_cond[, 1:6]
```

```
##              ld02      ld03      ld04      fl02      fl03      fl04
## AA      0.7833884 0.7521016 0.7284738 0.3228949 0.2560121 0.4560906
## Eriley  0.6954310 0.6565099 0.6299422 0.2333911 0.1774180 0.3236674
## Lf      0.7029431 0.6646121 0.6382397 0.2404182 0.1834251 0.3342160
## Nessman 0.6377067 0.5946209 0.5668688 0.1824325 0.1347627 0.2469403
## NWLF    0.7288502 0.6926410 0.6670184 0.2654522 0.2050578 0.3716382
## SPP     0.7936647 0.7633787 0.7401963 0.3345782 0.2665928 0.4729477
## Stevens 0.7186716 0.6816127 0.6556811 0.2554631 0.1963829 0.3567396
```

We display the average survival and flowering rates for each ancestral line.

```
rowMeans(preds_cond[, 1:3])
```

```
##        AA    Eriley        Lf   Nessman      NWLF       SPP   Stevens
## 0.7546546 0.6606277 0.6685983 0.5997321 0.6961699 0.7657466 0.6853218
```

```
rowMeans(preds_cond[, 4:6])
```

```
##        AA    Eriley        Lf   Nessman      NWLF       SPP   Stevens
## 0.3449992 0.2448255 0.2526864 0.1880451 0.2807161 0.3580396 0.2695285
```

We can compare with the estimates of expected Darwinian fitness (or the best surrogate of) for each ancestral line.

```
rowSums(preds_hdct)
```

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

Let's now compare with a zero-inflated Poisson model. The response for this model will be the sum of all head counts.

```
foo <- redata %>% filter(fit == 1) %>% group_by(id) %>%
  summarise(fitness = sum(resp), pop = unique(pop),
            ewloc = unique(ewloc), nsloc = unique(nsloc))
head(foo)
```

```
## # A tibble: 6 x 5
##      id fitness pop    ewloc nsloc
##   <int>   <int> <fct>  <int> <int>
## 1     1       0 NWLF      -8   -11
## 2     2       0 Eriley    -8   -10
## 3     3       0 NWLF      -8    -9
## 4     4       0 SPP       -8    -8
## 5     5       0 SPP       -8    -7
## 6     6       1 Eriley    -8    -6
```

Recall the `zeroinfl` function in the `pscl` package.

```
library(pscl)
m <- zeroinfl(fitness ~ pop + ewloc + nsloc, data = foo)
summary(m)
```

```
##
## Call:
## zeroinfl(formula = fitness ~ pop + ewloc + nsloc, data = foo)
##
## Pearson residuals:
##     Min      1Q  Median      3Q     Max
## -1.5892 -0.6795 -0.4511  0.4048  6.8134
##
## Count model coefficients (poisson with log link):
##              Estimate Std. Error z value Pr(>|z|)
## (Intercept)  1.556952   0.116514  13.363  < 2e-16 ***
## popEriley   -0.489889   0.143062  -3.424 0.000616 ***
## popLf       -0.446440   0.152620  -2.925 0.003443 **
## popNessman  -0.498853   0.228660  -2.182 0.029136 *
## popNWLF     -0.057281   0.129259  -0.443 0.657659
## popSPP      -0.117293   0.133304  -0.880 0.378917
## popStevens  -0.097473   0.139955  -0.696 0.486142
## ewloc        0.007642   0.005513   1.386 0.165711
## nsloc       -0.002746   0.005586  -0.492 0.622999
##
## Zero-inflation model coefficients (binomial with logit link):
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept) -0.03934    0.38506  -0.102  0.91862
## popEriley   -0.24266    0.45578  -0.532  0.59445
## popLf       -0.18248    0.48038  -0.380  0.70404
## popNessman   0.50797    0.58430   0.869  0.38465
## popNWLF      0.42221    0.42359   0.997  0.31889
## popSPP      -0.49195    0.46225  -1.064  0.28722
## popStevens   0.44164    0.44807   0.986  0.32430
## ewloc       -0.04561    0.01480  -3.082  0.00206 **
```

Estimates of fitness are nearly identical.

```
preds_hdct_0infl <- predict(m,
  newdata = data.frame(pop = pop, ewloc = 0, nsloc = 0),
  type = "response")

cbind(rowSums(preds_hdct), preds_hdct_0infl)
```

```
##                  preds_hdct_0infl
## AA       2.375984         2.418829
## Eriley   1.502068         1.657001
## Lf       1.565962         1.685632
## Nessman  1.064478         1.108977
## NWLF     1.800250         1.816442
## SPP      2.499468         2.657224
## Stevens  1.705529         1.724758
```

The `zeroinfl` does not come with a function for estimating standard errors of mean-value parameters. So we bootstrap.

```
library(parallel)
```

```
set.seed(13)
RNGkind("L'Ecuyer-CMRG")
nCores <- detectCores() - 2
B <- 1e4
system.time({out <- do.call(rbind, mclapply(1:B, mc.cores = nCores,
                                 function(j){
    m <- zeroinfl(fitness ~ pop + ewloc + nsloc,
                  data = foo[sample(1:nrow(foo), replace = TRUE), ])
    predict(m, newdata = data.frame(pop = pop, ewloc = 0, nsloc = 0),
            type = "response")
  })
)})
```

```
##    user  system elapsed
## 207.820  12.382  31.772
```

The aster model comes with useful lower standard errors.

```
cbind(preds_hdct_0infl, sqrt(diag(var(out))))
```

```
##   preds_hdct_0infl
## 1         2.418829 0.6467319
## 2         1.657001 0.2412353
## 3         1.685632 0.3226397
## 4         1.108977 0.3737391
## 5         1.816442 0.2457005
## 6         2.657224 0.3905494
## 7         1.724758 0.3279760
```

```
mean_value
```

```
##         estimates std. err.
## AA       2.375984 0.4460557
## Eriley   1.502068 0.1959073
## Lf       1.565962 0.2486880
## Nessman  1.064478 0.3090545
## NWLF     1.800250 0.1815734
## SPP      2.499468 0.2889009
## Stevens  1.705529 0.2216032
```

For completeness we now consider a parametric bootstrap.

```
set.seed(13)
RNGkind("L'Ecuyer-CMRG")
B <- 1e4
m <- zeroinfl(fitness ~ pop + ewloc + nsloc, data = foo)
pred_prob <- predict(m, type = "zero")
pred_lambda <- predict(m, type = "count")
system.time({out <- do.call(rbind, mclapply(1:B, mc.cores = nCores,
                                function(j){
  bar <- foo
  bar$resp <- rbinom(nrow(foo), size = 1, prob = pred_prob) *
    rpois(nrow(foo), lambda = pred_lambda)
  m <- zeroinfl(resp ~ pop + ewloc + nsloc, data = bar)
  predict(m, newdata = data.frame(pop = pop, ewloc = 0, nsloc = 0),
          type = "response")
  })
)})
```

```
##    user  system elapsed
## 207.861  12.708  31.955
```

The aster model comes with useful lower standard errors, but the parametric bootstrap procedure is more competitive.

```
cbind(preds_hdct_0infl, sqrt(diag(var(out))))
```

```
##   preds_hdct_0infl
## 1         2.418829 0.5260550
## 2         1.657001 0.1939094
## 3         1.685632 0.2469549
## 4         1.108977 0.3714930
## 5         1.816442 0.2380877
## 6         2.657224 0.2850335
## 7         1.724758 0.2932117
```

```
mean_value
```

```
##         estimates std. err.
## AA       2.375984 0.4460557
## Eriley   1.502068 0.1959073
## Lf       1.565962 0.2486880
## Nessman  1.064478 0.3090545
## NWLF     1.800250 0.1815734
## SPP      2.499468 0.2889009
## Stevens  1.705529 0.2216032
```