Supporting Data Analysis for "Unifying Life History Analysis for Inference of Fitness and Population Growth"

By

Ruth G. Shaw, Charles J. Geyer, Stuart Wagenius,
Helen H. Hangelbroek, and Julie R. Etterson

Technical Report No. 658
School of Statistics
University of Minnesota
July 7, 2007

Abstract

This technical report (TR) gives details of the data analyses backing up a paper (Shaw, et al., submitted) having the same authors as this TR and having the title that is quoted in the title of this TR There are three data sets. The first data set (Example 1 in the paper, Chapter 2 in this TR) is new, coming from Stuart Wagenius's group, and involves the perennial plant *Echinacea angustifolia* (narrow leaved purple coneflower). The second data set (Example 2 in the paper, Chapters 3 and 4 in this TR) has been previously analyzed (Etterson and Shaw, 2001; Etterson, 2004) and involves annual plant *Chamaecrista fasciculata* (partridge pea). The third data set (Example 3 in the paper, Chapter 5 in this TR) has been previously analyzed (Lenski and Service, 1982) and involves the insect *Uroleucon rudbeckiae* (brown ambrosia aphid).

All analyses are done in R (R Development Core Team, 2006), all using the `aster` contributed package, described by Geyer, et al. (2007) except for the analyses in the style of Lande and Arnold (1983), which use ordinary least squares regression. All analyses are done using the `Sweave` function in R, so that they are completely reproducible by anyone who has R with the `aster` package installed and LaTeX.

Chapter 1

Recreating this Document

1.1 Obtaining R and the Package

This document was created using the `aster` contributed package for the R statistical computing environment (R Development Core Team, 2006). It requires version 0.7-2 or later of the `aster` package, because that was the first version that includes the datasets used for the analyses in this document.

If the `aster` package has not yet been installed in your R installation, the R command

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

will do this. One can also do the equivalent using the GUI menus if on Apple Macintosh or Microsoft Windows. This may require root or administrator privileges.

After installation one issues the R command

```
> library(aster)
```

to use this package. One can also install the package in a nonstandard location (in one's home directory), but this requires changing the usage of the `library` function, and we do not explain this.

If the `aster` package has been installed in your R installation, but is not the current version on CRAN, the R command

```
update.packages("aster")
```

will upgrade to the current version.

If R has not been installed, follow the instructions on CRAN (`http://cran.r-project.org/`).

The version of R used to make this document is 2.5.0.

The version of the package used to make this document is 0.7-2.

1.2 Obtaining LaTeX

The `Sweave` command in R produces LaTeX output. To process it, you need the LaTeX document preparation system. If you are using Linux, this is probably just came with it. Free versions of LaTeX are also available for Apple Macintosh and Microsoft Windows.

1.3 Obtaining Files

Download from the aster web site `http:www.stat.umn.edu/geyer/aster` the following files

```
tr658.tex
start.Rnw
newnew.Rnw
chamae2.Rnw
chamae.Rnw
aphids.Rnw
chamae2-alpha.rda
chamae-alpha.rda
```

and put them all in the same directory ("folder" in GUI-speak).

1.4 Creating the Document

1.4.1 Sweave

Start R, make the director (folder) with the files the current working directory (there is a menu item on the R GUI for this) and use the `Sweave` command on each of the files with suffix `Rnw`, that is,

```
Sweave("start.Rnw")
Sweave("newnew.Rnw")
Sweave("chamae2.Rnw")
Sweave("chamae.Rnw")
Sweave("aphids.Rnw")
```

Some of these, especially `chamae.Rnw` take a while. Each "chunk" processed is reported so you can see something is happening, but there is at least one chunk that takes several minutes.

1.4.2 LaTeX

When all of these have been done, the R commands in the files with suffix `Rnw` have been executed and the results, both text output and image files containing plots have been produced. There will be new files with suffixes `tex`, `eps`, and `pdf`.

Now running the latex command on the top-level file `tr658.tex` will produce the document. On Linux just execute either of

```
latex tr658
pdflatex tr658
```

at the Linux command line to produce the document. It will be necessary to run these several times (until one no longer sees the message "Label(s) may have changed. Rerun to get cross-references right") to get all cross-references right.

1.4.3 Stangle

For those who do not want to mess with LaTeX, the `Stangle` function can be used instead of `Sweave`.

```
Stangle("newnew.Rnw")
Stangle("chamae2.Rnw")
Stangle("chamae.Rnw")
Stangle("aphids.Rnw")
```

will produce the files

```
newnew.R
chamae2.R
chamae.R
aphids.R
```

that contain only the R commands (the code "chunks") from the files with suffix `Rnw`. They can then be sourced, run in batch mode, whatever the user pleases.

1.4.4 The Role of the RDA Files

The two files having suffix `rda` are R data (RDA) files. One contains one "magic" number; the other contains two of them.

```
> rm(list = ls())
> load("chamae2-alpha.rda")
> ls()

[1] "alpha.fruit"

> alpha.fruit

[1] 2.46

> rm(list = ls())
> load("chamae-alpha.rda")
> ls()

[1] "alpha.fruit" "alpha.seed"

> alpha.fruit

[1] 2.48

> alpha.seed

[1] 16.18
```

These are shape parameters for negative binomial distributions used in Chapters 3 and 4. These RDA files are read near the beginning of these chapters. New values of these parameters are calculated by maximum likelihood in Sections 3.8 and 4.7 near the end of those chapters, and the new values are written out to the RDA files (clobbering the old values).

Thus these RDA files must exist in order to run `Sweave`. If one were to create these file with different numbers other than the ones provided, it might take several runs

```
Sweave("chamae2.Rnw")
Sweave("chamae.Rnw")
```

for the values written out at the end to converge to these values.

1.5 Reproducible Research

"Reproducible research" is a buzzphrase (Buckheit and Donoho, 1995; Gentleman and Temple Lang, 2004) that describes a simple basic idea. The following is quoted from the web page `http://www.stat.umn.edu/~charlie/Sweave`.

> It's the scientific ideal.
>
> - Research should be reproducible. Anything in a scientific paper should be reproducible by the reader.
> - Whatever may have been the case in low tech days, this ideal has long gone. Much scientific research in recent years is too complicated and the published details to scanty for anyone to reproduce it.
> - The lack of detail is not entirely the author's fault. Journals have severe page pressure and no room for full explanations.
> - For many years, the only hope of reproducibility is old-fashioned person-to-person contact. Write the authors, ask for data, code, whatever. Some authors help, some don't. If the authors are not cooperative, tough.
> - Even cooperative authors may be unable to help. If too much time has gone by and their archiving was not systematic enough and if their software was unportable, there may be no way to recreate the analysis.
> - Fortunately, the internet comes to the rescue. No page pressure there!
> - Nowadays, many scientific papers also point to supplementary materials on the internet, either at the journal's or the author's web site. It doesn't matter so long as the material is permanently available. Data, computer programs, whatever should be there.
>
> But even more, the entire analysis should be reproducible. In real science, this is hard. Redoing all the chemistry, or all the field work, or whatever is asking a lot.
>
> But in mathematical and computing sciences, like statistics, reproducibility is perfectly possible. It only takes will and knowledge to do it.

The R `Sweave` function, created by Friedrich Leisch (Leisch, 2002a,b), is very useful for reproducible research.

This technical report and the paper Shaw, et al. (submitted) are an example.

Chapter 2

Comparison of Fitness Among Groups

2.1 Introduction

Data were collected on *Echinacea angustifolia*. These data are in the `echin2` dataset in the `aster` contributed package to the R statistical computing environment. The data set is based on 557 Echinacea angustifolia plants that were planted as sprouts in a growth chamber. Seedling survivors were then transplanted to an experimental garden.

The components of fitness are the variables in the graphical model shown in Figure 2.1.

All response variables are collected in the response vector `resp` in the data frame `echin2`. Components of the response vector corresponding to the same individual have the same value of the `id` variable in `echin2`. Components of the response vector corresponding to the same node of the graphical model (to the same "original variable") have the same value of the `varb` variable in `echin2`. The levels of `varb`, the "original variables" are as follows.

Variables `lds`i measure survival of individuals in a growth chamber (periods are months). Variables `ld0`i measure survival of individuals in an experimental field plot after transplanting (periods are years). Variables `r0`i count number of rosettes (basal leaf clusters), which are a surrogate of fitness. The names use here are shortened from those in the dataset where they are `roct200`i. Individuals resulted from crosses in which (a) mates were from different remnant populations, (b) mates were chosen at random from the same remnant, or (c) mates shared their maternal parent (variable `crosstype`). Other variables in the data set measure location in the growth chamber (`flat`) or in the field plot (`posi` and `row`) and year of crossing (1999 or 2000, variable `yearcross`).

This data set was challenging because the covariate `flat` only makes sense in relation to response variables in the growth chamber (`lds`i) and the covariates `posi` and `row` only make sense in relation to response variables in the field plot (`ld0`i and `r0`i). The R formula mini-language is not designed to handle this sort of situation. Thus model matrices must be constructed "by hand."

Growth chamber is incorrectly referred to as "greenhouse" in the rest of this chapter.

$$\begin{tikzcd}
1 \arrow[r] & \texttt{lds1} \arrow[r] & \texttt{lds2} \arrow[r] & \texttt{lds3} \arrow[r] & \texttt{ld01} \arrow[r] & \texttt{ld02} \arrow[r] & \texttt{ld03} \arrow[r] \arrow[d] & \texttt{ld04} \arrow[r] \arrow[d] & \texttt{ld05} \arrow[d] \\
 & & & & & & \texttt{r03} & \texttt{r04} & \texttt{r05}
\end{tikzcd}$$

Figure 2.1: Graph for *Echinacea angustifolia* data. Arrows go from one life history component to another indicating conditional dependence in the aster model. Nodes are labeled by their associated variables. Root nodes are associated with the constant variable 1, indicating presence of individuals at the outset. If any parent variable is zero, then the child variable is also zero. Child variables are conditionally independent given the parent variable. If a parent variable is nonzero, then the conditional distribution of the child variable is as follows. `lds`i and `ld0`i are (conditionally) Bernoulli (zero indicates mortality, one indicates survival) and `r0`i is (conditionally) zero-truncated Poisson.

2.2 Data

Load the data. Look at number of variables, their names and types.

```
> library(aster)
> data(echin2)
> names(echin2)

[1] "crosstype" "yearcross" "flat"      "row"       "posi"
[6] "varb"      "resp"      "id"        "root"

> levels(echin2$varb)

 [1] "ld01"     "ld02"     "ld03"     "ld04"     "ld05"
 [6] "lds1"     "lds2"     "lds3"     "roct2003" "roct2004"
[11] "roct2005"
```

2.3 Set Up Aster Model

```
> vars <- c("lds1", "lds2", "lds3", "ld01", "ld02",
+     "ld03", "roct2003", "ld04", "roct2004", "ld05",
+     "roct2005")
> pred <- c(0, 1, 2, 3, 4, 5, 6, 6, 8, 8, 10)
> fam <- c(1, 1, 1, 1, 1, 1, 3, 1, 3, 1, 3)
> fam.default()[fam]

[[1]]
[1] "bernoulli"

[[2]]
[1] "bernoulli"
```

```
[[3]]
[1] "bernoulli"

[[4]]
[1] "bernoulli"

[[5]]
[1] "bernoulli"

[[6]]
[1] "bernoulli"

[[7]]
[1] "truncated.poisson(truncation = 0)"

[[8]]
[1] "bernoulli"

[[9]]
[1] "truncated.poisson(truncation = 0)"

[[10]]
[1] "bernoulli"

[[11]]
[1] "truncated.poisson(truncation = 0)"
```

Variable `vars` gives the names of the variables in the data that are components of the aster response vector. Variable `pred` gives the graphical model: `pred[i]` gives the index of the parent variable of variable `i` or zero if the parent is a root node. Variable `fam` specifies families. The original numeric code, now superseded, is 1 = Bernoulli, 2 = Poisson, 3 = zero-truncated Poisson. For backwards compatibility, these three models are still specified by the new function `fam.default`.

Count individuals and nodes

```
> nind <- length(unique(echin2$id))
> nnode <- length(levels(echin2$varb))
```

2.4 Hand Crafted Model Matrices

2.4.1 Aster Models

We construct model matrices (which are really three-way arrays in aster) by hand. The sad fact is that the R formula mini-language hardly qualifies as a language. It has minimal syntax and very little generality. It is just not up to specifying the models we want to fit.

However every aster model — a canonical affine model, as the accepted version of the *Biometrika* paper calls it — is specified by having an affine predictor of the form

$$\eta = a + M\beta \tag{$*$}$$

which is equation (8) in the paper with the left-hand side, which is φ in the paper, replaced by η, which is our notation for a general canonical parameter (either unconditional φ or conditional θ, as the case may be). In this document we are using unconditional models so we could have left (8) as it is in the paper with the canonical parameter denoted φ, but we strive for generality.

Equation ($*$) is a vector equation. Variable η is a vector whose length is the total number of nodes in the whole graph. The notion of the graph changed from the first draft of the paper to the third in response to the referee's comments. In the first draft (and this is what the `aster` package still implements). The graph specified by the vector `pred` has, call it `nnode` nodes. But this graph is repeated for, call it `nind` individuals. In the second and third drafts of the paper, individuals are invisible. The graph new sense is `nind` identical copies of the old-sense graph. Hence the new sense graph has `nind * nnode` nodes. The reasons for this change are two: there is no reason for the restriction to identical copies, the theory working perfectly well when individuals have different graphs and the combined graph is whatever it is, and the notation is simpler, the model matrices really being three-way arrays no longer being necessary (model matrices are really matrices in the version to appear in *Biometrika*, but the `aster` package is still stuck in the old notation).

Now ($*$), which is new sense, is a vector equation, so the dimension of each term must have dimension `nind * nnode`. That means η has this dimension, so does the known "origin" vector a, and so does the term $M\beta$, which is a matrix multiplication, M being a matrix, the *model matrix* and β being a vector of regression coefficients. Say the length of β is `ncoef`. Then the row dimension of M must be `nind * nnode` and the column dimension of M must be `ncoef`.

We do not have to worry about specifying the "origin" vector a. Usually M contains a in its range space and that means that the fitted mean value parameters do not depend on a, although the regression coefficients themselves are different, yet another reason regression coefficients are meaningless.

Our job is to construct the M that specifies the model we want. The R formula mini-language does this automagically in simple cases. In cases too complicated for the stupid computer to understand (and the R formula mini-language has only rudimentary knowledge of statistical modeling), we have to just do it ourselves.

After we have constructed the new sense model matrix M, then we must reshape it to use it with the `aster` function. With `nind`, `nnode`, and `ncoef` defined as above, the following code

```
mold <- array(as.numeric(mnew), c(nind, nnode, ncoef))
```

does this reshaping, turning the new sense model matrix `mnew`, which is really a matrix, into the old sense model matrix `mold`, which is really a three way array. For each `k`, the column `mnew[ , k]` of the new sense model matrix corresponding to one regression coefficient (one "predictor vector" in the regression jargon) corresponds to a matrix `mold[ , , k]` which has dimension `nind` by `nnode`.

Our strategy will be to work with new sense model matrices, since they are simpler, being really honest-to-God matrices, and since they correspond to the paper to appear anyway. Only at the end, just before they are fed into the `aster` function, will we reshape them to three-way arrays.

We also need to reshape the data (response and root) in the same fashion

```
> x <- echin2$resp
> dim(x) <- c(nind, nnode)
> r <- 0 * x + 1
```

2.4.2 Constructing One Model Matrix

Aster model matrices are just like any other model matrix used in regression. Theoretically they can be any matrix of the required dimension. In practice, they often have many columns that are zero-or-one valued. The columns that are zero-or-one valued are often called "dummy" predictor variables. Their effect is to add a constant, the corresponding regression coefficient β_k to the affine predictor η_i for the individuals having a one in m_{ik}. So this just puts in an additive term for individuals in a certain class (the class indicated by the dummy variable thought of as an indicator variable).

We have just one quantitative variable `posi`. Everything else is qualitative and corresponds to one or more dummy variables.

One Categorical Variable

We start building the model matrix for the largest model we will consider (call it the "supermodel") as follows.

```
> modmat.super <- NULL
> names.super <- NULL
> for (i in levels(echin2$varb)) {
+     modmat.super <- cbind(modmat.super, as.numeric(echin2$varb ==
+         i))
+     names.super <- c(names.super, i)
+ }
```

It is a standard S trick to start a recursion with an empty object `NULL`, which will act as a vector with no elements or a matrix with no columns. Each trip through the loop adds one column to the model matrix and a corresponding label to what will eventually be the column names for the model matrix. Note that we don't bother to construct regression coefficient labels that are exactly the same as those constructed by the R formula mini-language. Those are ridiculously verbose, done by a computer that is really very stupid. If anyone doesn't like these labels, they can just change the way `names.super` is defined.

Here we add one dummy variable for each element of

```
> levels(echin2$varb)
```

```
 [1] "ld01"     "ld02"     "ld03"     "ld04"     "ld05"
 [6] "lds1"     "lds2"     "lds3"     "roct2003" "roct2004"
[11] "roct2005"
```

what we usually call the response variables, but that terminology does not fit well with aster terminology, which says there are really `nrow(echin2)` or `nind * nnode` response variables. Whatever we call them, we have now added one dummy variable for each one of them. Thus each of these will have an independently fitted level (since each corresponds to a dummy variable).

A Definition

Next we define an indicator variable that indicates being in the greenhouse.

```
> in.greenhouse <- is.element(echin2$varb, grep("lds",
+     levels(echin2$varb), value = TRUE))
> print(unique(echin2$varb[in.greenhouse]), max.levels = 0)

[1] lds1 lds2 lds3

> print(unique(echin2$varb[!in.greenhouse]), max.levels = 0)

[1] ld01     ld02     ld03     roct2003 ld04     roct2004
[7] ld05     roct2005
```

Another Categorical Variable

Now we have another loop that adds one column per trip through the loop.

```
> for (i in levels(echin2$flat)) if (i > "1") {
+     modmat.super <- cbind(modmat.super, as.numeric(in.greenhouse &
+         echin2$flat == i))
+     names.super <- c(names.super, paste("flat", i,
+         sep = ""))
+ }
```

Here we add one column for each of

```
> levels(echin2$flat)

[1] "1" "2" "3"
```

that are greater than `"1"` in the sort order.

The reason for dropping one of the dummy variables is well known and taught in every regression class. The vector sum of all dummy variables corresponding to one categorical variable (in this case `echin2$varb`) is a vector of all ones, since each observation falls in exactly one category.

Since the same holds for every categorical variable, we must do one of the following in order to construct a model matrix that is not rank deficient.

- Drop one dummy variable from each group of dummy variables corresponding to one categorical variable. Then add a dummy variable corresponding to no categorical variable that is a column of all ones, a so-called “intercept” dummy variable.

- Drop one dummy variable from each group of dummy variables corresponding to one categorical variable, except for one group for which we keep them all.

Here we follow the latter strategy. We have no "intercept" and do not drop any of the dummy variables made in the first loop, but drop one from each loop thereafter.

Note that we do deal with the in-and-out-of-greenhouse issue explicitly and hence do what is obviously the Right Thing. The dummy variables we construct have a one if and only if the item in question has `in.greenhouse` equal to `TRUE` and `echin2$flat` equal to the value we are constructing a dummy variable for.

Yet Another Categorical Variable

Now we have another loop that adds one column per trip through the loop.

```
> for (i in levels(echin2$row)) if (i > "10") {
+     modmat.super <- cbind(modmat.super, as.numeric((!in.greenhouse) &
+         echin2$row == i))
+     names.super <- c(names.super, paste("row", i,
+         sep = ""))
+ }
```

Here we add one column for each of

```
> levels(echin2$row)

[1] "0"  "10" "11" "12" "13"
```

that are greater than `"10"` in the sort order.

This is very similar to the preceding loop except the level `"0"` is both bogus, a forlorn attempt to deal with the in-and-out of greenhouse issue. We must drop one dummy variable besides the bogus one.

Again we deal with the in-and-out-of-greenhouse issue explicitly and obviously do the Right Thing. The dummy variables we construct have a one if and only if the item in question has `in.greenhouse` equal to `FALSE` and `echin2$row` equal to the value we are constructing a dummy variable for.

And Another Categorical Variable

Now we have another loop that adds one column per trip through the loop.

```
> for (i in levels(echin2$yearcross)) if (i >= "2000") {
+     modmat.super <- cbind(modmat.super, as.numeric(echin2$yearcross ==
+         i))
+     names.super <- c(names.super, paste("yc", i,
+         sep = ""))
+ }
```

Here we add one column for each of

```
> levels(echin2$yearcross)

[1] "1999" "2000"
```

that are greater than or equal to "2000" in the sort order.

This is very similar to the preceding loops, but simpler. There is no bogus level to deal with and there is no in-and-out-of-greenhouse issue. The fact that the body of the if statement only executes once and we only add one dummy variable (one column of the model matrix) is not a problem. It just works.

Finally, a Quantitative Variable

No loop is needed to construct a predictor variable that is quantitative. We just make it a column of the model matrix.

```
> modmat.super <- cbind(modmat.super, as.numeric(!in.greenhouse) *
+     echin2$posi)
> names.super <- c(names.super, "posi")
```

We do have to deal with the in-and-out-of-greenhouse issue. The Right Thing is to make the quantitative variable zero when it has no effect, because that makes zero contribution to the canonical parameter, which is obviously what is wanted.

One might think that this "locates" all the in-greenhouse variables at `posi` equal to zero, but this is an illusion. The other dummy variables that in-greenhouse variables have give them independently fitted levels (corresponding to their regression coefficients), so there is no problem.

Interaction of Crosstype and Final Response in the Field

We have another loop that adds, again one column per trip through the loop.

```
> for (i in levels(echin2$crosstype)) if (i > "W") {
+     modmat.super <- cbind(modmat.super, as.numeric(echin2$crosstype ==
+         i & echin2$varb == "roct2005"))
+     names.super <- c(names.super, paste("cross",
+         i, sep = ""))
+ }
```

Here we add one column for each of

```
> levels(echin2$crosstype)

[1] "Br" "Wi" "Wr"
```

that are greater than "W" in the sort order.

Here we do something very tricky, only the same sort of trickiness involved in the `- pop` in the model formulae in the *Biometrika*, but even that was clear as mud, and when combined with the in-and-out-of-greenhouse issue, the R formula mini-language is just not up to the task.

The dummy variable(s) we add, 2 of them, have a one if and only if the item in question has `echin2$varb` equal to `"roct2005"` and `echin2$crosstype` equal to the value we are constructing a dummy variable for.

Interaction of Crosstype and Final Response in the Greenhouse

The columns we add here are just like those added in the preceding section except they involve the last greenhouse variable.

```
> for (i in levels(echin2$crosstype)) if (i > "W") {
+     modmat.super <- cbind(modmat.super, as.numeric(echin2$crosstype ==
+         i & echin2$varb == "lds3"))
+     names.super <- c(names.super, paste("crossgreen",
+         i, sep = ""))
+ }
```

2.4.3 Reshape This Matrix

```
> nodename <- unique(as.character(echin2$var))
> modmat.super <- array(as.vector(modmat.super), c(dim(x),
+     length(names.super)))
> dimnames(modmat.super) <- list(NULL, nodename, names.super)
```

The `array` function constructs arrays. We now assign appropriate dimnames using the `names.super` that we constructed as we constructed the matrix.

2.4.4 Extract Submatrices

Now we extract three submatrices of this supermodel matrix: one having only the field-crosstype effect, one having only the greenhouse-crosstype effect, and one having neither.

```
> ifield <- grep("crossW", names.super)
> names.super[ifield]

[1] "crossWi" "crossWr"

> igreen <- grep("crossgreenW", names.super)
> names.super[igreen]

[1] "crossgreenWi" "crossgreenWr"

> modmat.field <- modmat.super[, , -igreen]
> modmat.green <- modmat.super[, , -ifield]
> modmat.sub <- modmat.super[, , -c(ifield, igreen)]
```

2.5 Model Fits and Hypothesis Tests

2.5.1 Fit Models

Fit aster models. Although it is not obvious from the syntax, the function `aster` is generic with two methods `aster.formula` and `aster.default`. If the first argument is a formula the former is used. If not, as here, the latter is used.

```
> out.sub <- aster(x, r, pred, fam, modmat.sub)
> summary(out.sub)

Call:
NULL

         Estimate Std. Error z value Pr(>|z|)
ld01      0.79675    0.27149   2.935 0.003338 **
ld02      0.97698    0.24674   3.959 7.51e-05 ***
ld03      2.34798    0.37897   6.196 5.81e-10 ***
ld04      2.24919    0.39090   5.754 8.73e-09 ***
ld05      3.60101    0.27798  12.954  < 2e-16 ***
lds1     -0.55897    0.40883  -1.367 0.171554
lds2      0.90860    0.49505   1.835 0.066453 .
lds3      1.81132    0.39792   4.552 5.31e-06 ***
roct2003 -2.23956    0.20443 -10.955  < 2e-16 ***
roct2004 -1.13113    0.11786  -9.597  < 2e-16 ***
roct2005 -0.37779    0.08139  -4.642 3.45e-06 ***
flat2    -0.23666    0.13477  -1.756 0.079096 .
flat3     0.25855    0.18589   1.391 0.164262
row11     0.14098    0.03587   3.930 8.50e-05 ***
row12     0.11678    0.03450   3.385 0.000711 ***
row13     0.12177    0.03408   3.573 0.000353 ***
yc2000   -0.05429    0.02831  -1.918 0.055170 .
posi     -0.31281    0.06909  -4.527 5.97e-06 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1


> out.field <- aster(x, r, pred, fam, modmat.field)
> summary(out.field)

Call:
NULL

         Estimate Std. Error z value Pr(>|z|)
ld01      0.77621    0.27149   2.859 0.004248 **
ld02      0.95652    0.24676   3.876 0.000106 ***
ld03      2.32696    0.37898   6.140 8.25e-10 ***
ld04      2.22832    0.39098   5.699 1.20e-08 ***
ld05      3.67920    0.27981  13.149  < 2e-16 ***
lds1     -0.57777    0.40893  -1.413 0.157693
lds2      0.88916    0.49509   1.796 0.072498 .
lds3      1.79129    0.39796   4.501 6.76e-06 ***
roct2003 -2.25932    0.20455 -11.045  < 2e-16 ***
roct2004 -1.15086    0.11808  -9.747  < 2e-16 ***
```

```
roct2005 -0.28183    0.08575  -3.287 0.001014 **
flat2    -0.24194    0.13516  -1.790 0.073443 .
flat3     0.26025    0.18634   1.397 0.162510
row11     0.14506    0.03631   3.995 6.47e-05 ***
row12     0.12226    0.03528   3.465 0.000530 ***
row13     0.11932    0.03455   3.453 0.000554 ***
yc2000   -0.02960    0.02895  -1.022 0.306616
posi     -0.32096    0.06986  -4.594 4.34e-06 ***
crossWi  -0.65270    0.14628  -4.462 8.12e-06 ***
crossWr  -0.10687    0.13935  -0.767 0.443123
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

> out.green <- aster(x, r, pred, fam, modmat.green)
> summary(out.green)

Call:
NULL

             Estimate Std. Error z value Pr(>|z|)
ld01          0.78859    0.27151   2.904 0.003678 **
ld02          0.96886    0.24677   3.926 8.63e-05 ***
ld03          2.33927    0.37897   6.173 6.71e-10 ***
ld04          2.24076    0.39091   5.732 9.92e-09 ***
ld05          3.59308    0.27802  12.924  < 2e-16 ***
lds1         -0.55537    0.40941  -1.356 0.174941
lds2          0.90989    0.49531   1.837 0.066209 .
lds3          2.22018    0.43851   5.063 4.13e-07 ***
roct2003     -2.24693    0.20450 -10.987  < 2e-16 ***
roct2004     -1.13848    0.11799  -9.649  < 2e-16 ***
roct2005     -0.38512    0.08157  -4.721 2.34e-06 ***
flat2        -0.25543    0.13672  -1.868 0.061723 .
flat3         0.25410    0.18761   1.354 0.175598
row11         0.14202    0.03587   3.960 7.50e-05 ***
row12         0.11942    0.03469   3.442 0.000576 ***
row13         0.12159    0.03421   3.554 0.000380 ***
yc2000       -0.04521    0.02907  -1.555 0.119887
posi         -0.31388    0.06907  -4.545 5.50e-06 ***
crossgreenWi -1.01366    0.34051  -2.977 0.002912 **
crossgreenWr -0.39520    0.46967  -0.841 0.400099
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

> out.super <- aster(x, r, pred, fam, modmat.super)
> summary(out.super)
```

$$\begin{tikzcd}
& \texttt{out.super} \arrow[dl] \arrow[dr] & \\
\texttt{out.field} \arrow[dr] & & \texttt{out.green} \arrow[dl] \\
& \texttt{out.sub} &
\end{tikzcd}$$

Figure 2.2: Relationship of Aster Models. Arrows go from one model to another that is a nested submodel of it.

```
Call:
NULL

             Estimate Std. Error z value Pr(>|z|)
ld01          0.77412    0.27150   2.851 0.004355 **
ld02          0.95432    0.24678   3.867 0.000110 ***
ld03          2.32493    0.37898   6.135 8.53e-10 ***
ld04          2.22592    0.39099   5.693 1.25e-08 ***
ld05          3.65585    0.27983  13.064  < 2e-16 ***
lds1         -0.57400    0.40929  -1.402 0.160787
lds2          0.89143    0.49525   1.800 0.071867 .
lds3          2.04511    0.44247   4.622 3.80e-06 ***
roct2003     -2.26195    0.20461 -11.055  < 2e-16 ***
roct2004     -1.15350    0.11817  -9.761  < 2e-16 ***
roct2005     -0.29824    0.08701  -3.428 0.000609 ***
flat2        -0.25165    0.13647  -1.844 0.065188 .
flat3         0.25957    0.18745   1.385 0.166122
row11         0.14635    0.03641   4.019 5.85e-05 ***
row12         0.12414    0.03545   3.502 0.000462 ***
row13         0.12050    0.03470   3.472 0.000516 ***
yc2000       -0.02756    0.02940  -0.937 0.348565
posi         -0.32282    0.07003  -4.609 4.04e-06 ***
crossWi      -0.58086    0.15305  -3.795 0.000148 ***
crossWr      -0.08069    0.14552  -0.554 0.579272
crossgreenWi -0.52844    0.35997  -1.468 0.142100
crossgreenWr -0.29471    0.49249  -0.598 0.549563
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
```

2.5.2 Tests of Model Comparison

Figure 2.2 shows the relationship between these models. The only models that are not nested and so cannot be directly compared are `out.field` and `out.green` (although something can be said from the comparison of each to the models above and below).

The following `anova` commands do the four tests associated with the four arrows in Figure 2.2.

```
> anova(out.sub, out.field, out.super)

Analysis of Deviance Table

Model 1: (no formula)
Model 2: (no formula)
Model 3: (no formula)
  Model Df Model Dev Df Deviance P(>|Chi|)
1       18   2150.29
2       20   2127.54  2    22.75 1.146e-05
3       22   2125.36  2     2.18      0.34

> anova(out.sub, out.green, out.super)

Analysis of Deviance Table

Model 1: (no formula)
Model 2: (no formula)
Model 3: (no formula)
  Model Df Model Dev Df Deviance P(>|Chi|)
1       18   2150.29
2       20   2141.51  2     8.78      0.01
3       22   2125.36  2    16.15 0.0003114
```

We reach the following conclusions of a purely statistical nature (there is no point in scientific interpretations until the statistics is done).

First look at the right-hand side of Figure 2.2. We see that, although the model with crosstype effect in the greenhouse only (`out.green`) does fit significantly better ($P = 0.012$) than the baseline model with no crosstype effect, it does not fit as well ($P = 3.1 \times 10^{-4}$) as the supermodel with both crosstype effects. Thus looking at the right-hand side of Figure 2.2 only, the hypothesis tests of model comparison indicate that the supermodel (`out.super`) is the only model that fits the data.

Now we look at the left-hand side of Figure 2.2. We see that the model with crosstype effect in the field only (`out.field`) not only fits significantly better ($P = 1.1 \times 10^{-5}$) than the baseline model with no crosstype effect but also does not fit significantly worse ($P = 0.34$) than the supermodel with both crosstype effects. Thus looking at the left-hand side of Figure 2.2 only, the hypothesis tests of model comparison indicate that the middle model (`out.field`) is the most parsimonious model that fits the data (the super model also fits the data but no better than `out.field`).

Now looking at all the tests, the overall conclusion is that (`out.field`) is the most parsimonious model that fits the data.

2.6 Mean Value Parameters

2.6.1 In the Field

We need to make a new model matrix for hypothetical individuals having the same data structure as `modmat.super`. We want three individuals, one for each value of "cross." The easiest way to do this is to just use the first three "rows" (really layers of a three-way array) of `modmat.field` and adjust the predictors to be what we want.

```
> newmodmat.super <- modmat.super[1:3, , ]
> i <- grep("ld0[1-5]|lds[1-3]|roct200[3-5]", dimnames(newmodmat.super)[[3]])
> newmodmat.super[, , -i] <- 0
> newmodmat.super[2, "roct2005", "crossWi"] <- 1
> newmodmat.super[3, "roct2005", "crossWr"] <- 1
> newmodmat.super[2, "lds3", "crossgreenWi"] <- 1
> newmodmat.super[3, "lds3", "crossgreenWr"] <- 1
```

We set all "predictor" values except for those corresponding to response variables to zero (which we take to be a "neutral" or "typical" value) and then set the ones for cross type back to what we want them to be: individual 2 is cross type `"Wi"`, individual 3 is cross type `"Wr"`, and individual 1 is the remaining cross type `"Br"`.

Now we extract the other new model matrices just like we did the old.

```
> newmodmat.field <- newmodmat.super[, , -igreen]
> newmodmat.green <- newmodmat.super[, , -ifield]
> newmodmat.sub <- newmodmat.super[, , -c(ifield, igreen)]
```

Now we need root data and observed data of the appropriate shape to go with this model matrix, since the observed "x" data is ignored when we predict unconditional mean value parameters, we can use the same data structure for both.

```
> newroot <- array(1, dim = dim(newmodmat.field)[1:2])
```

We also need an `amat` argument that picks off the `"roct2005"` elements of the mean value parameter vector.

```
> amat <- array(0, dim = c(dim(newmodmat.field)[1:2],
+     3))
> for (i in 1:3) amat[i, dimnames(newmodmat.field)[[2]] ==
+     "roct2005", i] <- 1
```

This says we want the `i`-th component of the prediction to be the unconditional mean value parameter for `"roct2005"` for the `i`-th individual.

Now we can predict

```
> pout.super <- predict(out.super, newroot, newroot,
+     newmodmat.super, amat, se.fit = TRUE)
> pout.super$fit

[1] 0.9477683 0.5510009 0.8659104
```

```
> pout.field <- predict(out.field, newroot, newroot,
+     newmodmat.field, amat, se.fit = TRUE)
> pout.field$fit

[1] 0.9516398 0.5591503 0.8733273

> pout.green <- predict(out.green, newroot, newroot,
+     newmodmat.green, amat, se.fit = TRUE)
> pout.green$fit

[1] 0.8988469 0.8095031 0.8722694

> pout.sub <- predict(out.sub, newroot, newroot, newmodmat.sub,
+     amat, se.fit = TRUE)
> pout.sub$fit

[1] 0.8905837 0.8905837 0.8905837
```

Figure 2.3 is produced by the following code

```
> modelfits <- list(sub = pout.sub, green = pout.green,
+     field = pout.field, super = pout.super)

> conf.level <- 0.95
> crit <- qnorm((1 + conf.level)/2)
> crossnames <- levels(echin2$crosstype)
> icross <- c(1, 3, 2)
> foo <- 0.05
> modelcolors <- c("blue", "green", "red", "black")
> modellty <- c(3, 2, 1, 4)
> ylim <- NULL
> for (i in seq(along = modelfits)) {
+     thefit <- modelfits[[i]]
+     ylim <- range(ylim, thefit$fit + crit * thefit$se.fit)
+     ylim <- range(ylim, thefit$fit - crit * thefit$se.fit)
+ }
> xlim <- range(icross + 10 * foo, icross - 10 * foo)
> plot(NA, NA, xlim = xlim, ylim = ylim, xlab = "",
+     ylab = "", axes = FALSE)
> for (i in seq(along = modelfits)) {
+     thefit <- modelfits[[i]]
+     ytop <- thefit$fit + crit * thefit$se.fit
+     ybot <- thefit$fit - crit * thefit$se.fit
+     ymid <- thefit$fit
+     col <- modelcolors[i]
+     lty <- modellty[i]
+     jcross <- icross + (i - mean(seq(along = modelfits))) *
+         3 * foo
+     segments(jcross, ybot, jcross, ytop, lty = lty)
+     segments(jcross - foo, ybot, jcross + foo, ybot,
+         lty = lty)
+     segments(jcross - foo, ymid, jcross + foo, ymid,
+         lty = lty)
+     segments(jcross - foo, ytop, jcross + foo, ytop,
+         lty = lty)
+ }
> axis(side = 2)
> axis(side = 1, at = icross, labels = crossnames)
> title(xlab = "cross type")
```

and appears on p. 21.

2.6.2 In the Greenhouse

This section is very similar to the preceding. The only difference is that we want to predict `lds3` values rather than `roct2005` values.

We also need an `amat` argument that picks off the `"lds3"` elements of the mean value parameter vector.

```
> amat <- array(0, dim = c(dim(newmodmat.field)[1:2],
+     3))
> for (i in 1:3) amat[i, dimnames(newmodmat.field)[[2]] ==
+     "lds3", i] <- 1
```

Now we can predict as before

```
> pout.super <- predict(out.super, newroot, newroot,
+     newmodmat.super, amat, se.fit = TRUE)
> pout.super$fit

[1] 0.9281132 0.8242624 0.8987400

> pout.field <- predict(out.field, newroot, newroot,
+     newmodmat.field, amat, se.fit = TRUE)
> pout.field$fit

[1] 0.9126000 0.8583929 0.9036532

> pout.green <- predict(out.green, newroot, newroot,
+     newmodmat.green, amat, se.fit = TRUE)
> pout.green$fit

[1] 0.9371359 0.8439862 0.9094262

> pout.sub <- predict(out.sub, newroot, newroot, newmodmat.sub,
+     amat, se.fit = TRUE)
> pout.sub$fit
```

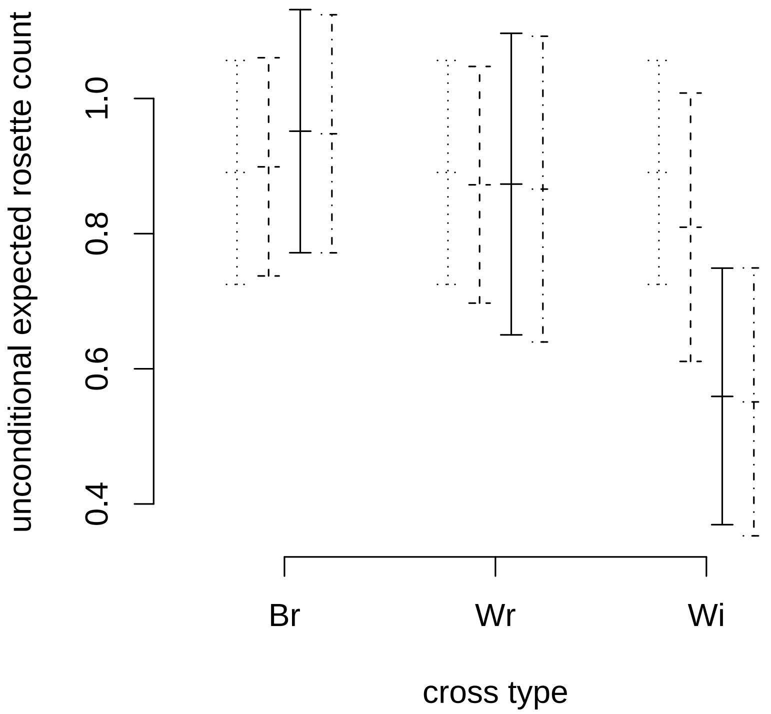

Figure 2.3: 95% confidence intervals for unconditional mean value parameter for fitness (rosette count in the last year recorded) for one "typical" individual for each cross type. Colors indicate model: blue, `model.sub`; green, `model.green`; red, `model.field`; black, `model.super`.

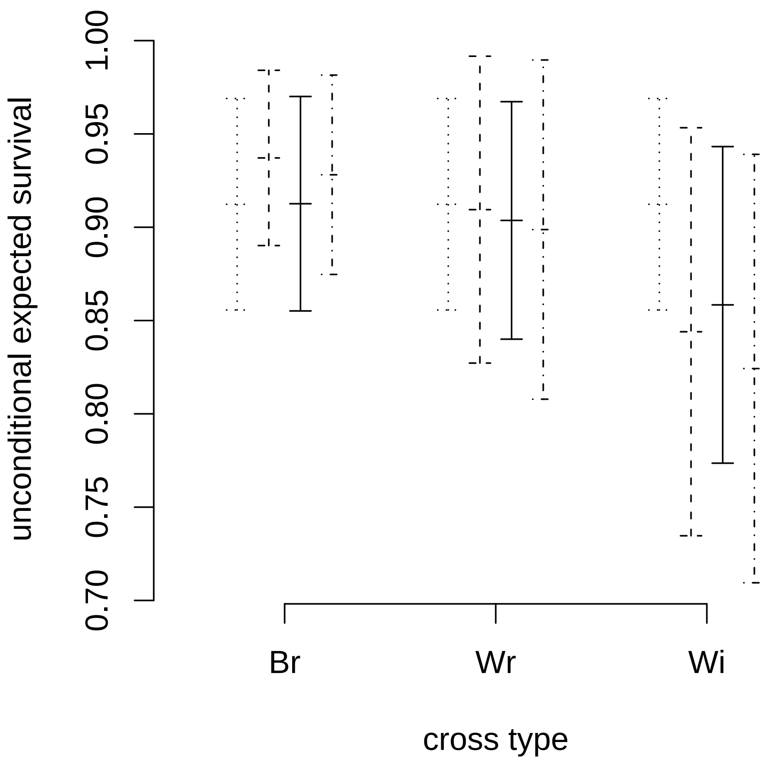

Figure 2.4: 95% confidence intervals for unconditional mean value parameter for the best surrogate for fitness measured in the greenhouse (survival in the last year recorded) for one "typical" individual for each cross type. Just as in Figure 2.3, colors indicate model: blue, `model.sub`; green, `model.green`; red, `model.field`; black, `model.super`.

```
[1] 0.9123147 0.9123147 0.9123147

> modelfits.field <- modelfits
> modelfits <- list(sub = pout.sub, green = pout.green,
+     field = pout.field, super = pout.super)
> modelfits.green <- modelfits
```

Figure 2.4 is now produced by the same code as Figure 2.3. and appears on p. 22.

Note that Figure 2.4 shows a similar pattern to Figure 2.3. Even though the mean value parameters predicted in Figure 2.3 correspond to canonical parameter values that contain the regression coefficients that differ between the two models and the mean value parameters predicted in Figure 2.4 do not.

So the failure of model `out.super` to be statistically significant does not mean that

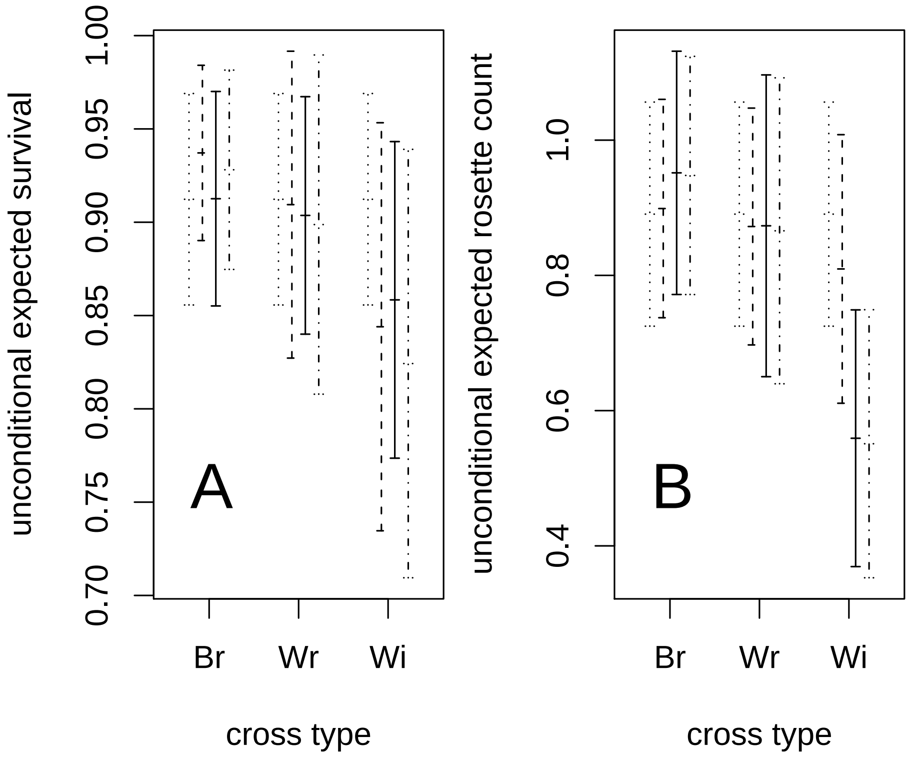

Figure 2.5: Plots for paper. Line types: dotted “sub” model, dashed “chamber” model, solid “field” model, and dot-dash “super” model.

there are no fitness effects in the greenhouse; it merely means that what effects there are are adequately explained by an unconditional aster model with no additional parameters.

2.6.3 Plot for Paper

Here we assemble the two preceding plots into one postscript file (Figure 2.5).

Chapter 3

Fitness Landscapes and Lande-Arnold Analysis using Aster Models

3.1 Introduction

Lande and Arnold (1983) proposed using the vector β of ordinary least squares (OLS) multiple regression coefficients (not including the intercept) from the regression of relative fitness on quantitative phenotypic characters as "a general solution to the problem of measuring the forces of directional selection acting directly on the characters" (Lande and Arnold, 1983, p. 1213).

Lande and Arnold actually give two different arguments justifying their procedure. The first argument, parts of which go back to Pearson (1903, cited by Lande and Arnold), involves a quantitative genetic model for the vector z of phenotypic characters. The second argument, which may have been independently invented by Lande and Arnold, but involves what statisticians call Stein's lemma (Stein, 1981), does not involve a quantitative genetic model, and is the one we follow here.

The assumptions made by Lande and Arnold (1983, p. 1213), somewhat generalized, are as follows.

(i) The vector of phenotypic characters z is multivariate normal.

(ii) The conditional expectation of relative fitness w given z, denote it

$$u(z) = E(w \mid z), \tag{3.1}$$

is a continuously differentiable function.

(iii) The expectation

$$\beta = E\{\nabla u(z)\} \tag{3.2}$$

exists.

(iv) The limits

$$\lim_{z_i \to \infty} u(z) f(z)$$
$$\lim_{z_i \to -\infty} u(z) f(z)$$

exist and are zero, letting each coordinate z_i go to plus or minus infinity, holding the rest fixed (Lande and Arnold, 1983, assume u is bounded, which is a sufficient condition for the limits above to be zero, but not a necessary condition).

Then it follows by integration by parts (Lande and Arnold, 1983, p. 1213) and from the definition of relative fitness, which requires $E(w) = 1$, that

$$\beta = E\{\nabla u(z)\} = \operatorname{var}(z)^{-1} \operatorname{cov}(w, z) \tag{3.3}$$

and for this reason Lande and Arnold call β the *directional selection gradient.* They also observe that the obvious empirical estimate of β, which is

$$\hat{\beta}_{\text{OLS}} = \widehat{\operatorname{var}}(z)^{-1} \widehat{\operatorname{cov}}(w, z), \tag{3.4}$$

where the hats on the var and cov operators denote the empirical analogs, is the ordinary least squares (OLS) regression estimate of w on z, done with intercept, which is not considered part of $\hat{\beta}$. They therefore seem to recommend (3.4) as an estimator of (3.3); this is not completely clear since they do not actually distinguish between (3.3) and (3.4), between the parameter β and its estimate $\hat{\beta}_{\text{OLS}}$. They do, however, use (3.4) as an estimator of (3.3) in their examples, and a large literature has followed them in this practice (Google Scholar, `http://scholar.google.com`, says "Cited by 816" as this is written).

Assumption (i) above, which we should also note in passing is also required by the "first argument" in Lande and Arnold (1983), implies that $\widehat{\operatorname{var}}(z)$ will be a good estimator of $\operatorname{var}(z)$, so long as the dimension of z is not too large relative to the amount of data. Assumptions (ii) through (iv) above are very weak. They do not even imply, for example, that $\operatorname{var}(w \mid z)$ is finite. They do imply, by the integration by parts argument, that $\operatorname{cov}(w, z)$ exists, which we should also note in passing is also required by the "first argument" in Lande and Arnold (1983), but this does not imply that $\widehat{\operatorname{cov}}(w, z)$ is a good estimator of it. It may be a very bad estimator, in which case (3.4) will be a very bad estimator of (3.3). In fact, this should be expected when the distribution of w is heavy tailed (since relative fitness w is nonnegative, this means heavy upper tail).

To start a search for better estimators, we make the following observation (which follows from the iterated conditional expectation theorem)

$$\operatorname{cov}(w, z) = \operatorname{cov}\bigl(u(z), z\bigr), \tag{3.5}$$

where $u(z)$ is defined by (3.1). This says that we can just as well use

$$\hat{\beta} = \widehat{\operatorname{var}}(z)^{-1} \widehat{\operatorname{cov}}\bigl(\hat{u}(z), z\bigr) \tag{3.6}$$

as an estimator of (3.3), where $\hat{u}(z)$ is any estimator of $u(z)$ that we can construct, such as, predicted mean values in any model we happen to have for the conditional distribution of w given z. If one uses a conventional linear, normal-theory, OLS regression model, then substituting (3.6) for (3.4) will do nothing (regressing OLS predicted values on predictors gives the same regression coefficients as the original regression), but there may be very large differences between (3.6) and (3.4) if one uses any other kind of regression model, for example, a generalized linear model (McCullagh and Nelder, 1989) or an aster model (Geyer, et al., 2007).

We make one more generalization, which is not new here, having been used in many examples in the literature (Mitchell-Olds and Shaw, 1987), but for which an adequate theoretical explanation has not yet been provided. Suppose we also have a vector of other variables e that we also wish to add to the regression. These may not be normally distributed, they may be categorical, for example, and may also be nonrandom (fixed by experimental design). Thus they cannot be considered part of the z in the Lande-Arnold argument (because z must be multivariate normal). They may also not be phenotypic characters. They may be environmental (hence our use of e to denote them), but they may also be just "other."

We apply the Lande-Arnold argument separately for each value of e. We assume

(i) The vector of phenotypic characters z is conditionally multivariate normal given e.

(ii) The conditional expectation of relative fitness w given z and e, denote it

$$u_e(z) = E(w \mid z, e) \tag{3.7}$$

is a continuously differentiable function of z.

(iii) The expectation

$$\beta(e) = E\{\nabla u_e(z) \mid e\} \tag{3.8}$$

exists for each value of e.

(iv) The limits

$$\lim_{z_i \to \infty} u_e(z) f(z)$$
$$\lim_{z_i \to -\infty} u_e(z) f(z)$$

exist and are zero, letting each coordinate z_i go to plus or minus infinity, holding the rest of the coordinates of z fixed and all coordinates of e fixed.

Then applying the Lande-Arnold argument for each value of e gives

$$\beta(e) = E\{\nabla u_e(z)\} = \operatorname{var}(z \mid e)^{-1} \operatorname{cov}(w, z \mid e) \tag{3.9}$$

This trivial extension of Lande-Arnold theory may not at first sight seem to do much. Typically, there will be far too many e values for $\beta(e)$ to be estimated naively using OLS regression to work. Since we have already seen that OLS regression may be a very bad idea for other reasons, this is no impediment. We proceed as everywhere else in statistics trying to find simple models for the gradient-vector-valued function (of "other" variables e) that is $\beta(e)$.

For example, the simplest model is that $\beta(e)$ is a constant function, which seems to get us back to using (3.6), but it does not. Recall that β or $\hat{\beta}$ uses OLS regression of something (preferably predicted mean values for a good model of the conditional distribution of w given z and e) on z and e and drops the intercept. Thus we model $\beta(e)$ as a constant function of e if we do OLS regression of something on z and e with only the intercept depending on e. Note that this allows the intercept term to depend on e in an arbitrarily complicated way. We just don't have "interaction" terms between z and e in the regression model. If we were going to write the kind of models just discussed in conventional terms, we would write

$$w = \alpha(e) + \beta^T z + \text{error}$$

where $\alpha(e)$ is an arbitrary function (of course, we don't believe the "+ error" part of this "model," which is given only to help the reader understand the structure of the assumed regression function). Of course we don't have to stop with this simplest model for $\beta(e)$, but it is interesting that even the simplest case of this theory (with e) is nontrivial.

When one uses the OLS regression estimator (3.4), the question arises about how to do statistical inference about β when the normality assumptions required for OLS F-tests are clearly violated. There is a large literature on this subject (see Mitchell-Olds and Shaw, 1987; Stanton and Thiede, 2005, and references cited therein). Transforming w to make it more normal, for example, doing OLS regression of $\log w$ on z is biologically wrong (Stanton and Thiede, 2005), destroying the logic of the Lande-Arnold theory which requires OLS regression of w (untransformed) on z. Lande and Arnold (1983) say that z may be transformed to make it more normal, hence more closely satisfy condition (i) above, but this may or may not make the distribution of w given (transformed) z more nearly satisfy OLS regression assumptions.

Sometimes, as in our example (Section 3.2 below), it is impossible to transform w to conditional normality given z. This happens, in particular, when a sizable fraction of individuals have (measured) fitness zero. The normal distribution is continuous. Any transformation of w has a large atom at whatever point zero is transformed to, hence is far from being continuous. The standard trick $\tilde{w} = \log(w + 1) - 1$, may make the distribution of $\tilde{w}$ have a more normal looking upper tail, but can do nothing about the fact that $\tilde{w}$ is nonnegative with a large atom at zero.

Having already decided to abandon OLS regression and use estimators of the form (3.6) or their analogs for estimating $\beta(e)$, which we have not provided equations for, we do not have the problem of depending on a procedure whose assumptions are badly violated.

If we can find an adequate model for the conditional distribution of w given z, we can simulate the sampling distribution of our estimator of β or $\beta(e)$ and use the simulations to make confidence intervals or do tests of significance. Strictly speaking, our simulation distribution depends on estimated parameters, hence is a parametric bootstrap. Our simulation will be valid if and only if our model actually is adequate and its estimated parameters are close enough to the true unknown parameters so that the simulation distribution is close to the true unknown distribution of w given z.

Mitchell-Olds and Shaw (1987) considered nonparametric bootstrap and jackknife methods in this situation, but the conditions for those procedures may be violated in many applications, particularly when fitness zero has appreciable probability. Mitchell-Olds and Shaw (1987) "wish to emphasize that resampling techniques are not a panacea" and cite literature cautioning about violations of required assumptions, in particular "both techniques are based on asymptotic theory (sample size must be 'large'), they assume residuals are identically and independently distributed ...." That last point is a killer in the current context. There is no generally accepted way to resample residuals for generalized linear models, much less for aster models. Resampling cases does not mimic the conditional distribution of response given predictors, which is what is crucial in this application.

3.2 Data

We reanalyze a subset of the data analyzed by Etterson and Shaw (2001). These data are in the `chamae2` dataset in the `aster` contributed package to the R statistical computing environment. This dataset is restricted to the Minnesota site of the original (larger) data.

These data are already in "long" format, no need to use the `reshape` function on them to do aster analysis. We will, however, need the "wide" format for Lande-Arnold analysis. So we do that, before making any changes (we will add newly defined variables) to `chamae2`.

```
> library(aster)
> data(chamae2)
> chamae2w <- reshape(chamae2, direction = "wide", timevar = "varb",
+     v.names = "resp", varying = list(levels(chamae2$varb)))
> names(chamae2w)

[1] "id"     "root"   "STG1N"  "LOGLVS" "LOGSLA" "BLK"    "fecund"
[8] "fruit"
```

Individuals of *Chamaecrista fasciculata* (common name, partridge pea) were obtained from three locations in the country and planted in three field sites. Of the complete data we only reanalyze here individuals from one site (Minnesota). For each individual, many characteristics were measured, three of which we consider phenotypic characters (so our z is three-dimensional), and others which combine to make up an estimate of fitness. The three phenotypic characters are reproductive stage (`STG1N`), log leaf number (`LOGLVS`), and log leaf thickness (`LOGSLA`). "At the natural end of the growing season, [they] recorded total pod number and seed counts from three representative pods; from these measures, [they] estimated [fitness]" (Etterson and Shaw, 2001, further explained in their note 12).

In this chapter we take fruit count as the measure of fitness because that gives the best example of aster analysis. (In Chapter 4 we combine fruit count and seed count, but the aster analysis is not as simple as in this chapter.)

We model fruit count as having a zero-inflated negative binomial distribution. The zero inflation allows for excess (or deficit) of individuals having zero fruit (over and above the small number of zeros that would occur if the distribution were pure negative binomial). In an aster model this is done by having a Bernoulli node followed by a zero-truncated negative binomial node (each individual having a simple graph with two nodes). This means the event that an individual has one or more fruits is modeled as Bernoulli, and the distribution of the number of fruit given that the number is at least one is modeled as zero-truncated negative binomial.

3.3 Aster Analysis

Then we set up the aster model framework.

```
> vars <- c("fecund", "fruit")
> pred <- c(0, 1)
```

We need to choose the non-exponential-family parameters (sizes) for the negative binomial distributions, since the `aster` package only does maximum likelihood for exponential family parameters. We start with the following values, which were chosen with knowledge of the maximum likelihood estimates for these parameters, which we find in Section 3.8.

```
> load("chamae2-alpha.rda")
> print(alpha.fruit)

[1] 2.46

> famlist <- list(fam.bernoulli(), fam.truncated.negative.binomial(size = alpha.fruit,
+     truncation = 0))
> fam <- c(1, 2)
```

We can now fit our first aster model.

```
> out1 <- aster(resp ~ varb + BLK, pred, fam, varb, id,
+     root, data = chamae2, famlist = famlist)
> summary(out1, show.graph = TRUE)

Call:
aster.formula(formula = resp ~ varb + BLK, pred = pred, fam = fam,
    varvar = varb, idvar = id, root = root, data = chamae2, famlist = famlist)



Graphical Model:
 variable predecessor
 fecund   root
 fruit    fecund
 family
 bernoulli
 truncated.negative.binomial(size = 2.46, truncation = 0)

              Estimate Std. Error z value Pr(>|z|)
(Intercept) -6.4692026  0.1354062 -47.776  < 2e-16 ***
varbfruit    7.4564500  0.1354514  55.049  < 2e-16 ***
BLK2        -0.0010528  0.0004303  -2.447   0.0144 *
BLK4         0.0021565  0.0003841   5.614 1.97e-08 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
```

The "response" `resp` is a numeric vector containing all the response variables (`fecund` and `fruit`). The "predictor" `varb` is a factor with two levels distinguishing with `resp` which original response variable an element is. The predictor `BLK` has not been mentioned so far. It is block within the field where the plants were grown.

Now we add phenotypic variables.

```
> out2 <- aster(resp ~ varb + BLK + LOGLVS + LOGSLA + STG1N,
+     pred, fam, varb, id, root, data = chamae2, famlist = famlist)
> summary(out2, info.tol = 1e-09)

Call:
aster.formula(formula = resp ~ varb + BLK + LOGLVS + LOGSLA +
    STG1N, pred = pred, fam = fam, varvar = varb, idvar = id,
    root = root, data = chamae2, famlist = famlist)

              Estimate Std. Error z value Pr(>|z|)
(Intercept) -6.0437702  0.1361498 -44.391  < 2e-16 ***
varbfruit    6.9883947  0.1364070  51.232  < 2e-16 ***
BLK2        -0.0022856  0.0004175  -5.475 4.37e-08 ***
BLK4        -0.0006200  0.0004041  -1.534  0.12498
LOGLVS       0.0158193  0.0004725  33.483  < 2e-16 ***
LOGSLA      -0.0066609  0.0024405  -2.729  0.00635 **
STG1N       -0.0011121  0.0001947  -5.711 1.12e-08 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

One might think we should use `varb * (LOGLVS + LOGSLA + STG1N)`.

```
> out2foo <- aster(resp ~ BLK + varb * (LOGLVS + LOGSLA +
+     STG1N), pred, fam, varb, id, root, data = chamae2,
+     famlist = famlist)
> summary(out2foo, info.tol = 1e-11)

Call:
aster.formula(formula = resp ~ BLK + varb * (LOGLVS + LOGSLA +
    STG1N), pred = pred, fam = fam, varvar = varb, idvar = id,
    root = root, data = chamae2, famlist = famlist)

                   Estimate Std. Error z value Pr(>|z|)
(Intercept)      -5.1621864  1.5413727  -3.349  0.00081 ***
BLK2             -0.0022692  0.0004155  -5.461 4.72e-08 ***
BLK4             -0.0006208  0.0004032  -1.540  0.12364
varbfruit         6.1067811  1.5418305   3.961 7.47e-05 ***
LOGLVS           -0.0669469  0.4954370  -0.135  0.89251
LOGSLA            1.9912512  1.8268008   1.090  0.27570
STG1N             0.3702374  0.1453418   2.547  0.01085 *
varbfruit:LOGLVS  0.0826889  0.4954705   0.167  0.86746
```

```
varbfruit:LOGSLA -1.9984472  1.8272876  -1.094  0.27410  
varbfruit:STG1N  -0.3714445  0.1453823  -2.555  0.01062 *
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
```

```
> feig <- eigen(out2foo$fisher, symmetric = TRUE, only.values = TRUE)$values
> range(feig)

[1] 1.060871e-01 3.717578e+09

> anova(out2, out2foo)

Analysis of Deviance Table

Model 1: resp ~ varb + BLK + LOGLVS + LOGSLA + STG1N
Model 2: resp ~ BLK + varb * (LOGLVS + LOGSLA + STG1N)
  Model Df Model Dev Df Deviance P(>|Chi|)
1        7     57461                        
2       10     57453  3        8   0.04293
```

It turns out the Fisher information is nearly singular, as shown by the need for the `info.tol = 1e-11` argument to the `summary` command and also by the eigenvalues of the Fisher information matrix, the condition number (ratio of largest and smallest eigenvalues) being 3.5×10^{10}. Thus we do not use this model.

An alternative model with the same number of parameters as `out2` puts in the regression coefficients only at the "fitness" level (here `fruit`). This is similar to the example in Geyer, et al. (2007). Because we are fitting an unconditional aster model, the effects of these terms are passed down to `fecund`.

```
> foo <- as.numeric(as.character(chamae2$varb) == "fruit")
> chamae2$LOGLVSfr <- chamae2$LOGLVS * foo
> chamae2$LOGSLAfr <- chamae2$LOGSLA * foo
> chamae2$STG1Nfr <- chamae2$STG1N * foo
> out6 <- aster(resp ~ varb + BLK + LOGLVSfr + LOGSLAfr +
+     STG1Nfr, pred, fam, varb, id, root, data = chamae2,
+     famlist = famlist)
> summary(out6, info.tol = 1e-09)

Call:
aster.formula(formula = resp ~ varb + BLK + LOGLVSfr + LOGSLAfr + 
    STG1Nfr, pred = pred, fam = fam, varvar = varb, idvar = id, 
    root = root, data = chamae2, famlist = famlist)

              Estimate Std. Error z value Pr(>|z|)    
(Intercept) -6.0045602  0.1363337 -44.043  < 2e-16 ***
varbfruit    6.9491773  0.1366290  50.862  < 2e-16 ***
BLK2        -0.0022851  0.0004174  -5.474 4.39e-08 ***
```

```
BLK4         -0.0006202  0.0004041  -1.535  0.12480
LOGLVSfr      0.0158197  0.0004725  33.482  < 2e-16 ***
LOGSLAfr     -0.0066704  0.0024411  -2.733  0.00628 **
STG1Nfr      -0.0011126  0.0001948  -5.712 1.12e-08 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
```

It is not possible to compare `out2` and `out6` by standard methods (likelihood ratio test) because the models are not nested. They seem to fit equally well, and `out6` more directly models the relation of fitness (here defined as `fruit`) to phenotypic variables.

In both `out2` and `out6` the regression coefficient for the phenotypic variable `STG1N` is statistically significant ($P = 1.1 \times 10^{-8}$ and $P = 1.1 \times 10^{-8}$, respectively, although one should treat with extreme caution any P-values printed out when the `info.tol` argument is used in the `summary` call).

We now want to model the canonical parameter corresponding to fitness as a quadratic function of phenotype variables. Since `STG1N` is quite discrete, taking only six theoretically possible values with the vast majority of individuals having only two of these,

```
> sort(unique(chamae2w$STG1N))

[1] 1 2 3

> tabulate(chamae2w$STG1N)

[1] 1094  228  917
```

it does not make sense to model fitness as a quadratic function of `STG1N`. Thus we drop this phenotypic variable from the analysis.

```
> out7 <- aster(resp ~ varb + BLK + LOGLVSfr + LOGSLAfr,
+     pred, fam, varb, id, root, data = chamae2, famlist = famlist)
> summary(out7)

Call:
aster.formula(formula = resp ~ varb + BLK + LOGLVSfr + LOGSLAfr,
    pred = pred, fam = fam, varvar = varb, idvar = id, root = root,
    data = chamae2, famlist = famlist)

              Estimate Std. Error z value Pr(>|z|)
(Intercept) -6.0099069  0.1364221 -44.054  < 2e-16 ***
varbfruit    6.9486443  0.1367514  50.812  < 2e-16 ***
BLK2        -0.0022225  0.0004176  -5.322 1.03e-07 ***
BLK4        -0.0010662  0.0003982  -2.677  0.00742 **
LOGLVSfr     0.0174130  0.0003987  43.673  < 2e-16 ***
LOGSLAfr    -0.0066010  0.0024390  -2.706  0.00680 **
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
```

Now we add quadratic terms, looking for a maximum of the fitness function.

```
> out8 <- aster(resp ~ varb + BLK + LOGLVSfr + LOGSLAfr +
+     I(LOGLVSfr^2) + I(LOGSLAfr^2) + I(2 * LOGLVSfr *
+     LOGSLAfr), pred, fam, varb, id, root, data = chamae2,
+     famlist = famlist)
> summary(out8, info.tol = 1e-09)

Call:
aster.formula(formula = resp ~ varb + BLK + LOGLVSfr + LOGSLAfr +
    I(LOGLVSfr^2) + I(LOGSLAfr^2) + I(2 * LOGLVSfr * LOGSLAfr),
    pred = pred, fam = fam, varvar = varb, idvar = id, root = root,
    data = chamae2, famlist = famlist)

                             Estimate Std. Error z value Pr(>|z|)
(Intercept)                -5.5424746  0.1458750 -37.995  < 2e-16 ***
varbfruit                   6.2100415  0.1552610  39.997  < 2e-16 ***
BLK2                       -0.0024726  0.0004085  -6.053 1.42e-09 ***
BLK4                       -0.0005549  0.0003721  -1.491   0.1359
LOGLVSfr                    0.1569760  0.0151565  10.357  < 2e-16 ***
LOGSLAfr                   -0.2405565  0.0534496  -4.501 6.78e-06 ***
I(LOGLVSfr^2)              -0.0229378  0.0024441  -9.385  < 2e-16 ***
I(LOGSLAfr^2)              -0.1251870  0.0303839  -4.120 3.79e-05 ***
I(2 * LOGLVSfr * LOGSLAfr)  0.0103318  0.0059896   1.725   0.0845 .
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

> anova(out7, out8)

Analysis of Deviance Table

Model 1: resp ~ varb + BLK + LOGLVSfr + LOGSLAfr
Model 2: resp ~ varb + BLK + LOGLVSfr + LOGSLAfr + I(LOGLVSfr^2) + I(LOGSLAfr^2) +
  Model Df Model Dev Df Deviance P(>|Chi|)
1        6     57495
2        9     57296  3      199 5.771e-43
```

The quadratic term is highly significant. We could try more experimentation with different models, but we deem this one satisfactory and base our analysis on this (`out8`). See Section 3.7 below for residual analysis.

3.4 The Regression Function for Fitness

In this section we examine the regression function $u_e(z)$ that corresponds to the model. For simplicity, we answer a slightly different question: what is $E(W \mid z, e)$ changing from w (relative fitness) to W (raw fitness). Since $E(w \mid z, e)$ and $E(W \mid z, e)$ are proportional, they have maxima, minima, or saddle points (as the case may be) in the same place.

By assumption (in this chapter) W is the variable `fruit`, a canonical statistic of the aster model. Thus its conditional expectation is the corresponding mean value parameter (τ in the notation of Geyer, et al., 2007). Since the map from canonical parameter (φ in the notation of Geyer, et al., 2007) is monotone increasing, it maps maxima to maxima, minima to minima, and saddle points to saddle points (see Section 3.10 for details). Thus we can look at φ as a function of the phenotypic predictor variables (which is quadratic) to see which type of stationary point we have (if we have any, which we must because a quadratic function always has stationary points) and where the stationary point is.

The regression function for canonical parameter, ignoring intercept terms (which do not affect the location or type of stationary point and which depend on the environmental or "other" predictors e), is quadratic, letting φ_1 denote the canonical parameter for `LOGLVSfr` and φ_2 denote the canonical parameter for `LOGSLAfr`, it is of the form

$$\mathbf{a}^T \boldsymbol{\varphi} + \boldsymbol{\varphi}^T \mathbf{A} \boldsymbol{\varphi} \tag{3.10}$$

where $\mathbf{a}$ is a vector and $\mathbf{A}$ a matrix defined as follows.

```
> a1 <- out8$coefficients["LOGLVSfr"]
> a2 <- out8$coefficients["LOGSLAfr"]
> a <- c(a1, a2)
> A11 <- out8$coefficients["I(LOGLVSfr^2)"]
> A22 <- out8$coefficients["I(LOGSLAfr^2)"]
> A12 <- out8$coefficients["I(2 * LOGLVSfr * LOGSLAfr)"]
> A <- matrix(c(A11, A12, A12, A22), 2, 2)
```

Since the eigenvalues of $\mathbf{A}$ are all negative

```
> eigen(A, symmetric = TRUE, only.values = TRUE)$values

[1] -0.02190421 -0.12622051
```

the regression function has a (unique) maximum. The derivative of (3.10) being

$$\mathbf{a}^T + 2\boldsymbol{\varphi}^T \mathbf{A}$$

the maximum is achieved at the point

$$-\tfrac{1}{2}\mathbf{A}^{-1}\mathbf{a}$$

which is computed in R by

```
> max8 <- (-solve(A, a)/2)
> print(max8)

  LOGLVSfr   LOGSLAfr
 3.1044207 -0.7045769
```

Figure 3.1 (page 36) shows the scatterplot of data values for `LOGLVS` and `LOGSLA` and the contours of the estimated quadratic fitness function (3.10). It is made by the following code.

```
> plot(chamae2w$LOGLVS, chamae2w$LOGSLA, xlab = "LN", ylab = "SLA")
> ufoo <- par("usr")
> nx <- 101
> ny <- 101
> z <- matrix(NA, nx, ny)
> x <- seq(ufoo[1], ufoo[2], length = nx)
> y <- seq(ufoo[3], ufoo[4], length = ny)
> points(max8[1], max8[2], pch = 19)
> for (i in 1:nx) {
+     for (j in 1:ny) {
+         b <- c(x[i], y[j])
+         z[i, j] <- sum(a * b) + as.numeric(t(b) %*% A %*%
+             b)
+     }
+ }
> b <- as.numeric(max8)
> contour(x, y, z, add = TRUE)
> contour(x, y, z, levels = c(0.325), add = TRUE)
```

3.5 Lande-Arnold Analysis

In contrast to the aster analysis, the Lande-Arnold analysis is very simple.

```
> chamae2w$relfit <- chamae2w$fruit/mean(chamae2w$fruit)
> lout1 <- lm(relfit ~ LOGLVS + LOGSLA + STG1N, data = chamae2w)
> summary(lout1)

Call:
lm(formula = relfit ~ LOGLVS + LOGSLA + STG1N, data = chamae2w)

Residuals:
     Min       1Q   Median       3Q      Max 
-1.98030 -0.38051 -0.05526  0.30614  4.70941 

Coefficients:
            Estimate Std. Error t value Pr(>|t|)    
(Intercept) -2.79228    0.18755 -14.888  < 2e-16 ***
LOGLVS       1.64621    0.05592  29.439  < 2e-16 ***
LOGSLA       0.14274    0.19445   0.734    0.463    
STG1N       -0.10732    0.01470  -7.302 3.92e-13 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 0.6428 on 2235 degrees of freedom
Multiple R-Squared: 0.3245,        Adjusted R-squared: 0.3236 
F-statistic:   358 on 3 and 2235 DF,  p-value: < 2.2e-16
```

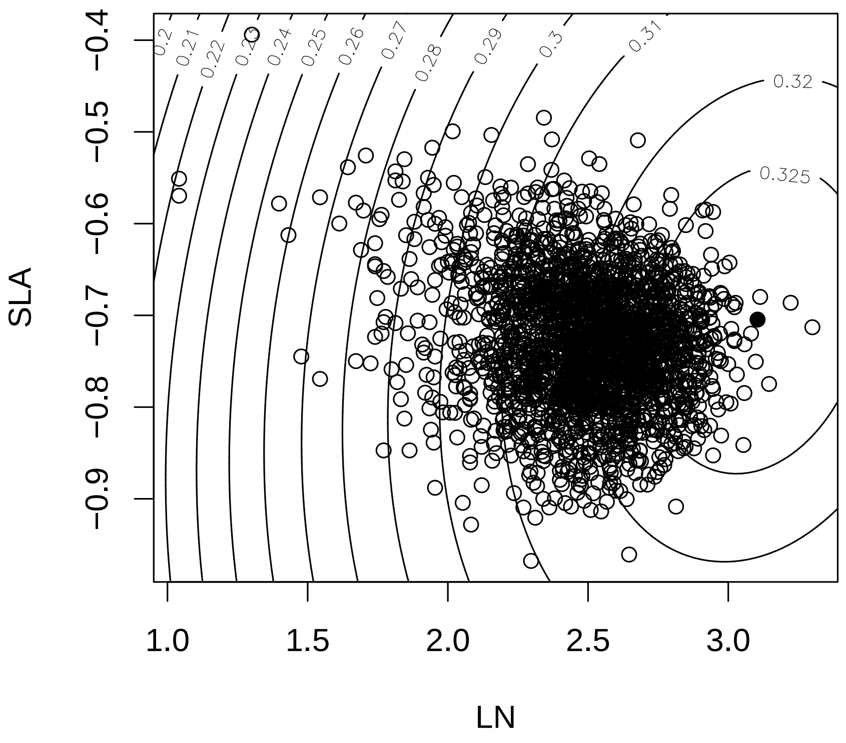

Figure 3.1: Scatterplot of `LOGLVS` versus `LOGSLA` with contours of the estimated quadratic fitness function. Variable names in axis labels changes to `LN` and `SLA`, respectively to agree with paper, which used these names. Solid dot is the point where the estimated fitness function achieves its maximum. Note "z" coordinate is only a monotone function of fitness (3.10), not actual fitness (compare Figure 3.7).

The information contained in the printout of `summary(lout1)` with the exception of the `Estimate` column is unreliable because the OLS model assumptions are not satisfied, as acknowledged by Etterson and Shaw (2001) and Etterson (2004). Therefore measures of statistical significance including standard errors (`Std. Error` column), t-statistics (`t value` column), and P-values (`Pr(>|t|)` column) are erroneous.

Also the `(Intercept)` regression is of no interest (not part of β or γ).

We can also estimate $\beta(e)$ as a constant function, where e is `BLK`, our comments about that applying to OLS regression estimates as well as to (our as yet to be determined "better" estimates).

```
> lout2 <- lm(relfit ~ BLK + LOGLVS + LOGSLA + STG1N, data = chamae2w)
> summary(lout2)

Call:
lm(formula = relfit ~ BLK + LOGLVS + LOGSLA + STG1N, data = chamae2w)

Residuals:
     Min       1Q   Median       3Q      Max
-1.82819 -0.35898 -0.05095  0.28502  4.60265

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept) -2.73021    0.18460 -14.790  < 2e-16 ***
BLK2        -0.22484    0.03313  -6.787 1.46e-11 ***
BLK4         0.04761    0.03468   1.373     0.17
LOGLVS       1.62216    0.05699  28.462  < 2e-16 ***
LOGSLA       0.00494    0.19411   0.025     0.98
STG1N       -0.12909    0.01488  -8.674  < 2e-16 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 0.6322 on 2233 degrees of freedom
Multiple R-Squared: 0.3472,        Adjusted R-squared: 0.3457
F-statistic: 237.5 on 5 and 2233 DF,  p-value: < 2.2e-16
```

Note that if we paid attention to the putative P-values from the OLS regression (which we should not because the assumptions for OLS do not hold) we would make the wrong decision about which phenotypic variable to drop from the analysis (dropping `LOGSLA` and keeping `STG1N`).

Note also that there is no reason why predictors that are important for the regression function itself are important for its average gradient (even assuming that the assumptions for Stein's lemma hold). Thus there is no reason why our aster analysis should agree with the Lande-Arnold analysis. When the analyses disagree, our approach, which directly estimates the regression function $u(z)$ or $u_e(z)$ itself, has more direct evolutionary implications.

Lande and Arnold (1983) also gave a justification for quadratic regression with an appeal to integration by parts (Stein's lemma). Let us try that, using the same predictors as for our aster model `out8` so that we get a direct comparison.

```
> lout8 <- lm(relfit ~ BLK + LOGLVS + LOGSLA + I(LOGLVS^2) +
+     I(LOGSLA^2) + I(2 * LOGLVS * LOGSLA), data = chamae2w)
> summary(lout8)

Call:
lm(formula = relfit ~ BLK + LOGLVS + LOGSLA + I(LOGLVS^2) + I(LOGSLA^2) +
    I(2 * LOGLVS * LOGSLA), data = chamae2w)

Residuals:
     Min       1Q   Median       3Q      Max
-2.22330 -0.34023 -0.06447  0.28152  3.74871

Coefficients:
                        Estimate Std. Error t value Pr(>|t|)
(Intercept)              0.19353    1.25205   0.155 0.877177
BLK2                    -0.21240    0.03285  -6.466 1.23e-10 ***
BLK4                    -0.03398    0.03382  -1.005 0.315125
LOGLVS                  -4.33814    0.63377  -6.845 9.85e-12 ***
LOGSLA                 -10.46591    2.84477  -3.679 0.000240 ***
I(LOGLVS^2)              1.28399    0.13081   9.816  < 2e-16 ***
I(LOGSLA^2)             -6.77004    1.93748  -3.494 0.000485 ***
I(2 * LOGLVS * LOGSLA)   0.09666    0.36360   0.266 0.790391
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 0.6267 on 2231 degrees of freedom
Multiple R-Squared: 0.359,       Adjusted R-squared: 0.357
F-statistic: 178.5 on 7 and 2231 DF,  p-value: < 2.2e-16
```

Note that the Lande-Arnold analysis seems to have gotten the wrong sign for the coefficient of `I(LOGLVS^2)`, where “wrong” means disagreeing with the aster analysis. Again, this is no surprise. The OLS model is not actually trying to fit the regression surface so there is no surprise that it tells us little about the regression surface and what it does say is misleading. Since we cannot tell whether a matrix is positive definite or indefinite by looking at its coefficients, we must form the matrix of coefficients of quadratic terms and look at its eigenvalues to see what we really have.

```
> a1 <- lout8$coefficients["LOGLVS"]
> a2 <- lout8$coefficients["LOGSLA"]
> olsa <- c(a1, a2)
> A11 <- lout8$coefficients["I(LOGLVS^2)"]
> A22 <- lout8$coefficients["I(LOGSLA^2)"]
> A12 <- lout8$coefficients["I(2 * LOGLVS * LOGSLA)"]
> olsA <- matrix(c(A11, A12, A12, A22), 2, 2)
```

Since the eigenvalues of $\mathbf{A}$ are all negative

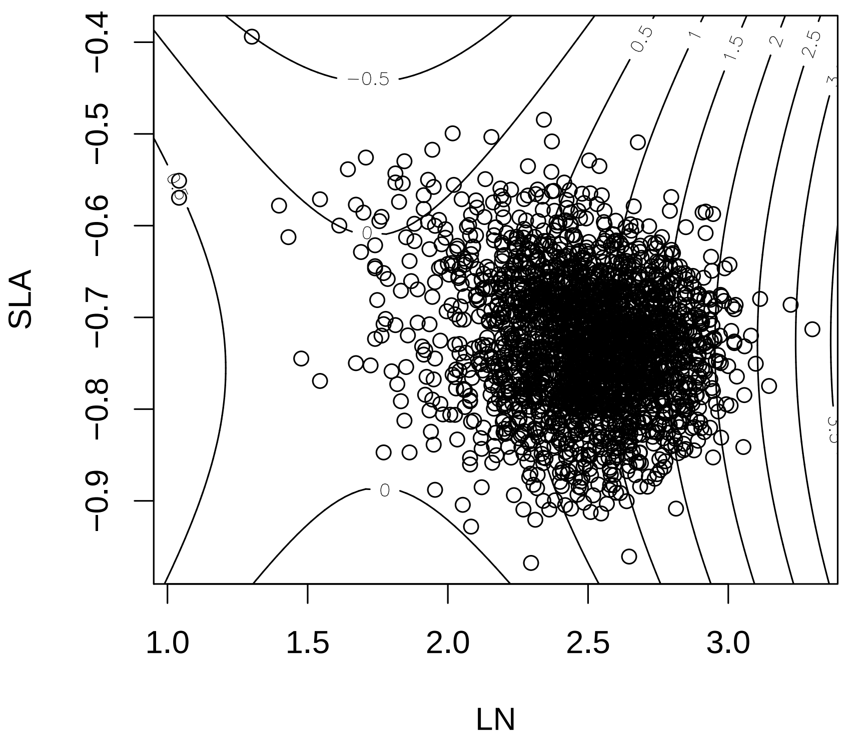

Figure 3.2: Scatterplot of `LOGLVS` versus `LOGSLA` with contours of the quadratic fitness function estimated by OLS. As in Figure 3.1 variable names in axis labels changes to `LN` and `SLA`, respectively to agree with paper, which used these names. This surface does not have a maximum, so, unlike Figure 3.1 no maximum is shown.

```
> eigen(olsA, symmetric = TRUE, only.values = TRUE)$values
```

```
[1]  1.285148 -6.771197
```

Since one eigenvalue is positive and the other negative the OLS regression suggests dispersive selection in one direction and stabilizing selection in another. Of course, since the assumptions for OLS regression are not met, there is no reason to trust this “suggestion”.

Figure 3.2 (page 39) is just like Figure 3.1 except it shows the quadratic fitness function estimated by OLS (the Lande-Arnold analysis) rather than by aster.

3.6 More on Quadratic Lande-Arnold Analysis

With $u(z)$ defined by (3.1), Lande and Arnold (1983) defined

$$\beta = E\{\nabla u(z)\} \tag{3.11a}$$

$$\gamma = E\{\nabla^2 u(z)\} \tag{3.11b}$$

calling (3.11a) the *directional selection gradient* and (3.11b) the *stabilizing selection gradient*. Equation (3.2) just repeats (3.2) above. Note that β is a vector and γ is a matrix.

Because β and γ are averages over the assumed multivariate normal distribution of z, it may not make sense to draw plots like our Figure 3.2. In fact, such a figure is the fitness landscape $u(z)$ only if $\nabla^2 u(z)$ is constant (not, in fact, a function of z) and the fitness landscape is actually quadratic.

Lande and Arnold (1983) show that, if z is jointly mean-zero multivariate normal, then β and γ can be estimated by the linear and quadratic regression coefficients in the OLS regression of relative fitness on z. This depends crucially on z being jointly multivariate normal. In our data, z is the two-dimensional vector with components `LOGLVS` and `LOGSLA`.

For our data (and for most data sets), this multivariate normality assumption is not correct, although perhaps not too badly violated for these two variables. The scatterplot of these variables is the pattern of dots in Figure 3.1 or 3.2 and although not elliptical is not too far from that.

One technical issue that we ignored, is that Lande and Arnold (1983) require that predictor variables be centered, so that (again assuming joint multivariate normality) the estimates for β will be the same whether or not the quadratic term is added. Thus we try this.

```
> chamae2w$cLOGLVS <- chamae2w$LOGLVS - mean(chamae2w$LOGLVS)
> chamae2w$cLOGSLA <- chamae2w$LOGSLA - mean(chamae2w$LOGSLA)
> lout1too <- lm(relfit ~ BLK + cLOGLVS + cLOGSLA, data = chamae2w)
> summary(lout1too)

Call:
lm(formula = relfit ~ BLK + cLOGLVS + cLOGSLA, data = chamae2w)

Residuals:
     Min       1Q   Median       3Q      Max 
-1.82071 -0.39027 -0.07706  0.29520  4.66456 

Coefficients:
            Estimate Std. Error t value Pr(>|t|)    
(Intercept)  1.07834    0.02430  44.368  < 2e-16 ***
BLK2        -0.21916    0.03367  -6.510 9.27e-11 ***
BLK4        -0.01086    0.03458  -0.314    0.754    
cLOGLVS      1.74174    0.05621  30.984  < 2e-16 ***
cLOGSLA      0.07791    0.19713   0.395    0.693    
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
```

```
Residual standard error: 0.6426 on 2234 degrees of freedom
Multiple R-Squared: 0.3252,       Adjusted R-squared: 0.324
F-statistic: 269.1 on 4 and 2234 DF,  p-value: < 2.2e-16

> lout1took <- lm(relfit ~ BLK + LOGLVS + LOGSLA, data = chamae2w)
> summary(lout1took)

Call:
lm(formula = relfit ~ BLK + LOGLVS + LOGSLA, data = chamae2w)

Residuals:
     Min       1Q   Median       3Q      Max
-1.82071 -0.39027 -0.07706  0.29520  4.66456

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept) -3.20534    0.17919 -17.887  < 2e-16 ***
BLK2        -0.21916    0.03367  -6.510 9.27e-11 ***
BLK4        -0.01086    0.03458  -0.314    0.754
LOGLVS       1.74174    0.05621  30.984  < 2e-16 ***
LOGSLA       0.07791    0.19713   0.395    0.693
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 0.6426 on 2234 degrees of freedom
Multiple R-Squared: 0.3252,       Adjusted R-squared: 0.324
F-statistic: 269.1 on 4 and 2234 DF,  p-value: < 2.2e-16

> as.numeric(coefficients(lout1too)[4:5])

[1] 1.74174277 0.07791206

> as.numeric(coefficients(lout1took)[4:5])

[1] 1.74174277 0.07791206
```

Note that centering does not change the estimates of β.

Now for the quadratic regression.

```
> lout8too <- lm(relfit ~ BLK + cLOGLVS + cLOGSLA + I(cLOGLVS^2) +
+     I(cLOGSLA^2) + I(2 * cLOGLVS * cLOGSLA), data = chamae2w)
> summary(lout8too)

Call:
lm(formula = relfit ~ BLK + cLOGLVS + cLOGSLA + I(cLOGLVS^2) +
    I(cLOGSLA^2) + I(2 * cLOGLVS * cLOGSLA), data = chamae2w)
```

```
Residuals:
     Min       1Q   Median       3Q      Max 
-2.22330 -0.34023 -0.06447  0.28152  3.74871 

Coefficients:
                         Estimate Std. Error t value Pr(>|t|)    
(Intercept)               1.03697    0.02660  38.983  < 2e-16 ***
BLK2                     -0.21240    0.03285  -6.466 1.23e-10 ***
BLK4                     -0.03398    0.03382  -1.005 0.315125    
cLOGLVS                   1.92049    0.05933  32.368  < 2e-16 ***
cLOGSLA                  -0.12717    0.19782  -0.643 0.520380    
I(cLOGLVS^2)              1.28399    0.13081   9.816  < 2e-16 ***
I(cLOGSLA^2)             -6.77004    1.93748  -3.494 0.000485 ***
I(2 * cLOGLVS * cLOGSLA)  0.09666    0.36360   0.266 0.790391    
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1 

Residual standard error: 0.6267 on 2231 degrees of freedom
Multiple R-Squared: 0.359,       Adjusted R-squared: 0.357 
F-statistic: 178.5 on 7 and 2231 DF,  p-value: < 2.2e-16 
```

```
> as.numeric(coefficients(lout8too)[6:8])

[1]  1.28398842 -6.77003697  0.09665758

> as.numeric(coefficients(lout8)[6:8])

[1]  1.28398842 -6.77003697  0.09665758
```

Note that centering does not change the estimates of γ.

However, the estimates of β do change depending on whether the quadratic terms are in or out.

```
> as.numeric(coefficients(lout1too)[4:5])

[1] 1.74174277 0.07791206

> as.numeric(coefficients(lout8too)[4:5])

[1]  1.9204918 -0.1271687
```

They do not change much, but they do change. We may regard this as a failure of the multivariate normality assumption about the phenotype vector, since Lande and Arnold (1983) state that the coefficients would not change if z were multivariate normal.

3.7 Goodness of Fit

In this section we examine goodness of fit to the assumed conditional distributions for `fruit` given `fecund == 1` by looking at a residual plot.

Residual analysis of generalized linear models (GLM) is tricky. (Our aster model becomes a GLM when we consider only the conditional distribution associated with one arrow.) Many different residuals have been proposed Davison and Snell (1991). We start with the simplest, so called Pearson residuals.

```
> xi.hat <- predict(out8, model.type = "cond", parm.type = "mean")
> xi.hat <- matrix(xi.hat, nrow = nrow(out8$x), ncol = ncol(out8$x))
> theta.hat <- predict(out8, model.type = "cond", parm.type = "canon")
> theta.hat <- matrix(theta.hat, nrow = nrow(out8$x), ncol = ncol(out8$x))

> woof <- chamae2w$fruit[chamae2w$fecund == 1]
> range(woof)

[1]    1 1390

> nwoof <- length(woof)
> woof.theta <- theta.hat[chamae2w$fecund == 1, 2]
> woof.xi <- xi.hat[chamae2w$fecund == 1, 2]
> wgrad <- double(nwoof)
> winfo <- double(nwoof)
> for (i in 1:nwoof) {
+     wgrad[i] <- famfun(famlist[[2]], deriv = 1, woof.theta[i])
+     winfo[i] <- famfun(famlist[[2]], deriv = 2, woof.theta[i])
+ }
> all.equal(woof.xi, wgrad)

[1] TRUE

> pearson <- (woof - woof.xi)/sqrt(winfo)
```

Figure 3.3 (page 44) shows the scatter plot of the Pearson residuals for fruit count plotted against the expected fruit count given that fruit count is nonzero (for each individual) for individuals with nonzero fitness only.

3.8 Maximum Likelihood Estimation of Size

The `aster` function does not calculate the correct likelihood when the size parameters are considered unknown, because it drops terms that do not involve the exponential family parameters. However, the full log likelihood is easily calculated in R.

```
> x <- out8$x
> logl <- function(alpha.fruit, theta, x) {
+     x.fecund <- x[, 1]
```

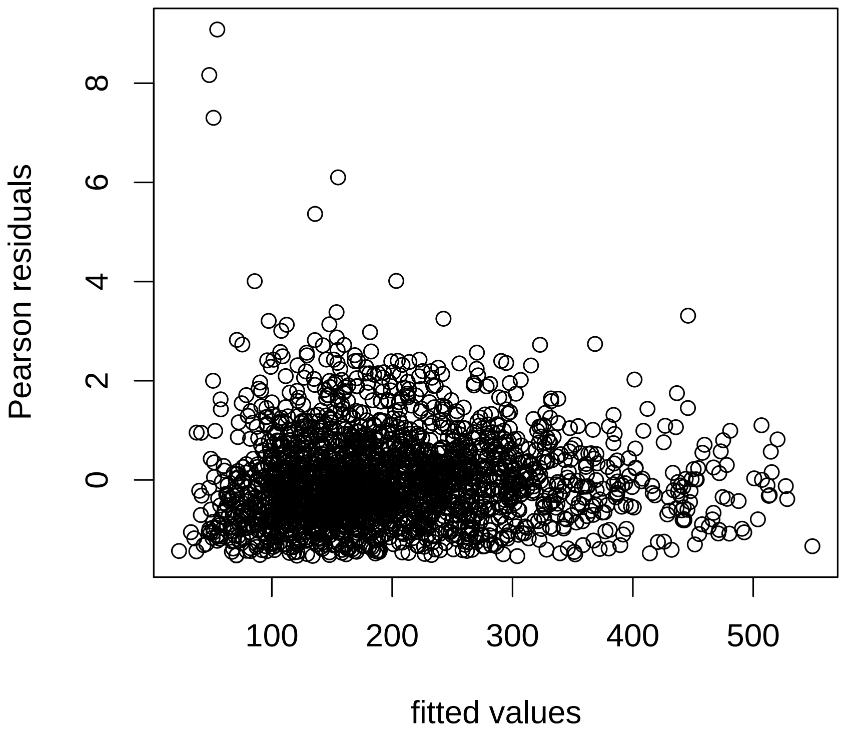

Figure 3.3: Pearson residuals for fruit count given nonzero fitness plotted against fitted values.

```
+     theta.fecund <- theta[, 1]
+     p.fecund <- 1/(1 + exp(-theta.fecund))
+     logl.fecund <- sum(dbinom(x.fecund, 1, p.fecund,
+         log = TRUE))
+     x.fruit <- x[x.fecund == 1, 2]
+     theta.fruit <- theta[x.fecund == 1, 2]
+     p.fruit <- (-expm1(theta.fruit))
+     logl.fruit <- sum(dnbinom(x.fruit, size = alpha.fruit,
+         prob = p.fruit, log = TRUE) - pnbinom(0, size = alpha.fruit,
+         prob = p.fruit, lower.tail = FALSE, log = TRUE))
+     logl.fecund + logl.fruit
+ }
```

We then calculate the profile likelihood for the size parameter alpha.fruit maximizing over the other parameters, evaluating the profile log likelihood on a grid of points.

```
> alpha.fruit.seq <- seq(1.5, 4.5, 0.25)
> logl.seq <- double(length(alpha.fruit.seq))
> for (i in 1:length(alpha.fruit.seq)) {
+     famlist.seq <- famlist
+     famlist.seq[[2]] <- fam.truncated.negative.binomial(size = alpha.fruit.seq[i],
+         truncation = 0)
+     out8.seq <- aster(out8$formula, pred, fam, varb,
+         id, root, data = chamae2, famlist = famlist.seq,
+         parm = out8$coefficients)
+     theta.seq <- predict(out8.seq, model.type = "cond",
+         parm.type = "canon")
+     dim(theta.seq) <- dim(x)
+     logl.seq[i] <- logl(alpha.fruit.seq[i], theta.seq,
+         x)
+ }
> alpha.foo <- seq(min(alpha.fruit.seq), max(alpha.fruit.seq),
+     0.01)
> logl.foo <- spline(alpha.fruit.seq, logl.seq, n = length(alpha.foo))$y
> imax <- seq(along = alpha.foo)[logl.foo == max(logl.foo)]
> alpha.fruit <- alpha.foo[imax]
> save(alpha.fruit, file = "chamae2-alpha.rda", ascii = TRUE)
```

At the end of this chunk we save the maximum likelihood estimates in a file which is read in at the beginning of this chapter.

Figure 3.4 (page 46) shows the profile log likelihood for the size parameters.

3.9 OLS Diagnostic Plots

Although unnecessary because we know the assumptions justifying OLS are badly violated, here are some diagnostic plots for the OLS regression.

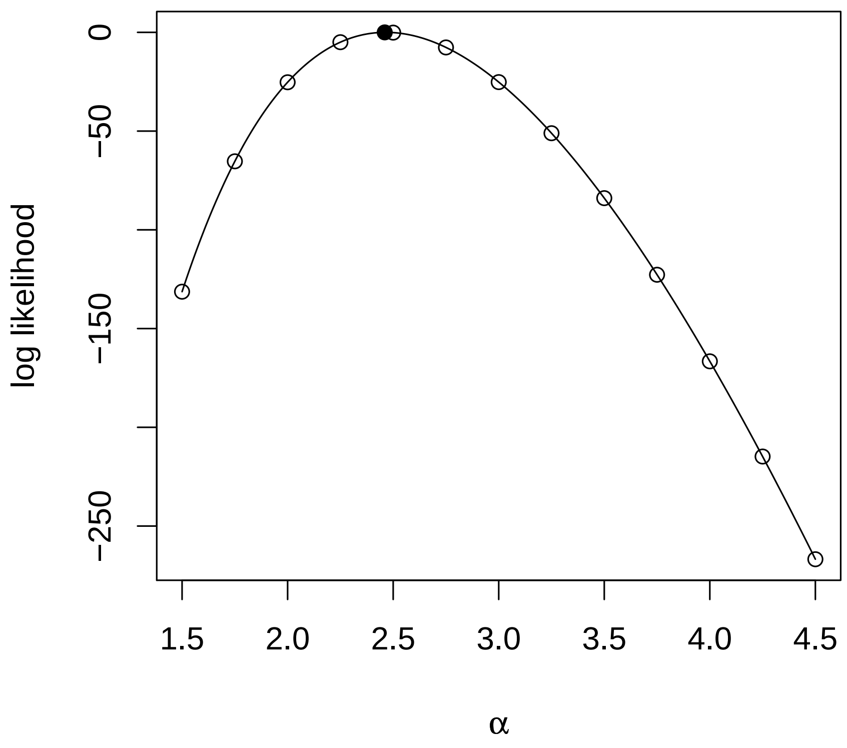

Figure 3.4: Profile log likelihood for size parameter for the (zero-truncated) negative binomial distribution of fruit. Hollow dots are points at which the log likelihood was evaluated exactly. Curve is the interpolating spline. Solid dot is maximum likelihood estimate.

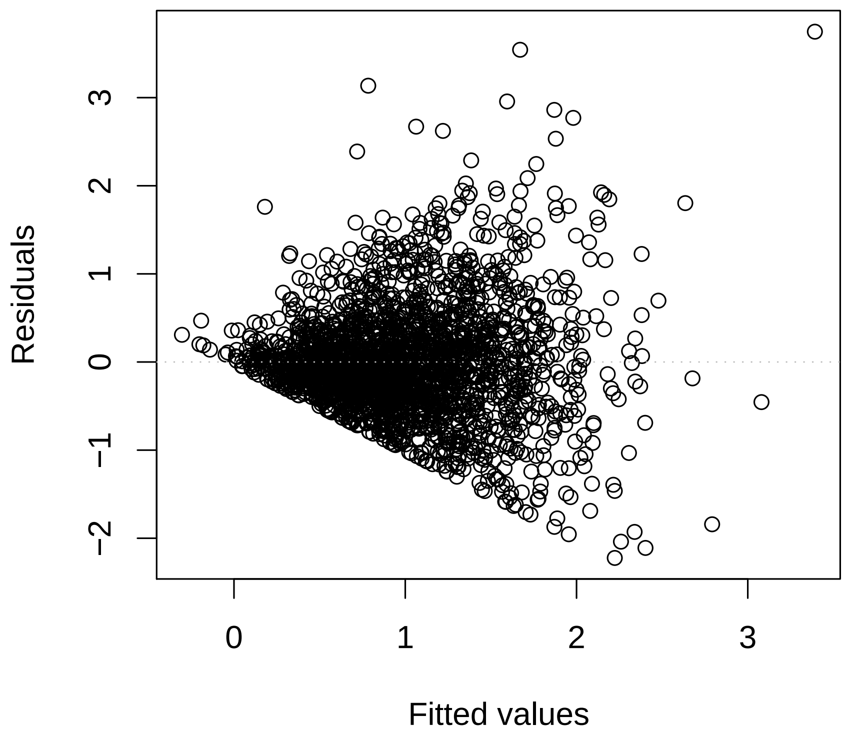

Figure 3.5: Residuals versus Fitted plot for OLS fit with blocks.

Figure 3.5 (page 47) shows the plot of residuals versus fitted values made by the R statement

```
> plot(lout8, which = 1, add.smooth = FALSE, id.n = 0,
+     sub.caption = "", caption = "")
```

Figure 3.6 (page 48) shows the Normal Q-Q (quantile-quantile) plot made by the R statement

```
> plot(lout8, which = 2, id.n = 0, sub.caption = "")
```

Clearly the errors are highly non-normal

3.10 Monotone Transformation of Fitness

This section makes explicit the monotonicity argument used on p. 34 that allows us to look at the quadratic surface shown in Figure 3.1 rather than the fitness surface (landscape) itself.

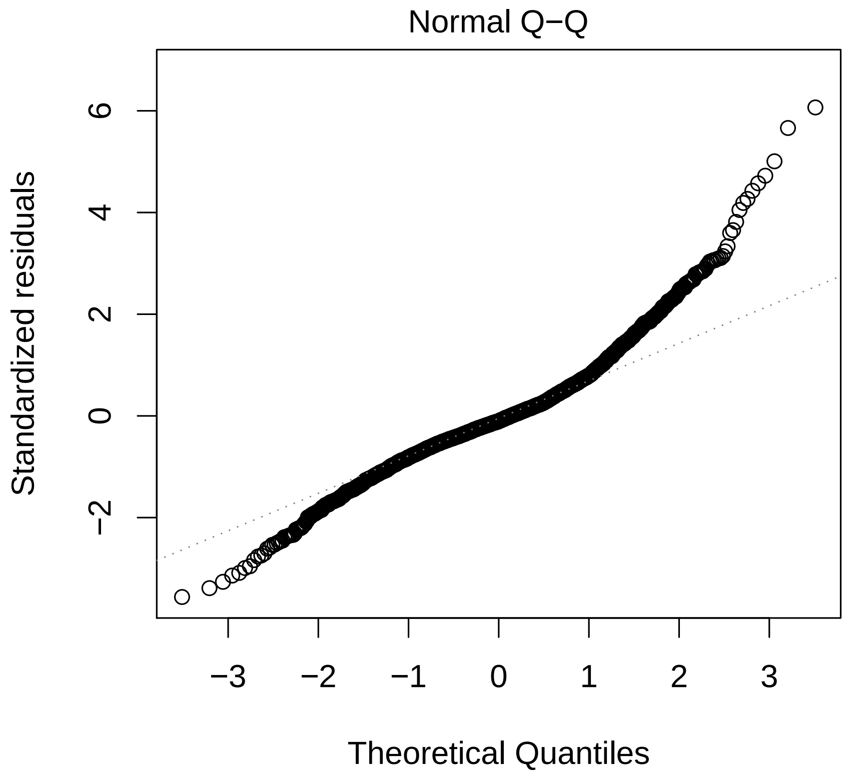

Figure 3.6: Normal Q-Q plot for OLS fit with blocks.

This argument holds whenever an unconditional aster model is used and one component of the canonical statistic is taken to be observed fitness and the corresponding canonical parameter the only parameter taken to be a function of phenotypic predictor variables z.

The argument is very simple but a bit confusing since it applies to a aster model that is not the actual model used. Consider any unconditional aster model. Then

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

gives the relation between the unconditional canonical parameter vector φ and the vector of regression coefficients β and

$$Y = M^T X$$

gives the relation between the original data vector X and the canonical statistic vector Y (which is the minimal sufficient statistic). These are (8) and (9) in Geyer, et al. (2007).

We are supposing that one component of Y, say Y_k is observed fitness Let

$$w = E_\beta(Y_k)$$

be expected fitness. Then the relationship between expected fitness and the corresponding canonical parameter β_k is given by (22) in Geyer, et al. (2007)

$$\frac{\partial w}{\beta_k} = \frac{\partial E_\beta(Y_k)}{\partial \beta_k} > 0$$

which means that, other components of β being held fixed, w is a strictly increasing function of β_k.

Now suppose that we make β_k and no other component of β a function of phenotypic predictor variables z. Then by the argument given above

$$w(z) = E_{\beta(z)}(Y_k)$$

is a strictly increasing function of $\beta_k(z)$ other predictor variables being held fixed. Hence $w(z)$ has a maximum where $\beta_k(z)$ has a maximum, and so forth.

That is the end of the argument, but there is still the tricky part to deal with. When we actually fit an aster model, we cannot have regression coefficients that are arbitrary functions of predictors. We have to write them as linear functions of other coefficients. So in our example (p. 34) we actually had

$$\beta_k(z) = \beta_{k1} z_1 + \beta_{k2} z_2 + \beta_{k11} z_1^2 + 2\beta_{k12} z_1 z_2 + \beta_{k22} z_2^2$$

So what is one regression coefficient β_k in our argument becomes several regression coefficients in the actual model.

So let us repeat the argument keeping track of what is fitness in the actual example. There fitness is fruit count. Therefore our argument requires that phenotypic predictors only directly affect fruit count (only involve fruit count variables), and this is what we have done by creating the predictor variables `LOGLVSfr` and `LOGSLAfr`.

3.11 Plotting the Fitness Landscape

If one does not want to use the argument of the preceding section or if one wants to see the actual (modeled) fitness surface rather than a plot like Figure 3.1, which is only a monotone transformation of the fitness surface, this section shows how to do that.

Figure 3.7 (page 51) shows the scatterplot of data values for `LOGLVS` and `LOGSLA` and the contours of the estimated fitness function that is the transform of (3.10) to the mean value parameter scale. It is made by the following code.

```
> plot(chamae2w$LOGLVS, chamae2w$LOGSLA, xlab = "LN", ylab = "SLA")
> points(max8[1], max8[2], pch = 19)
> ufoo <- par("usr")
> nx <- 101
> ny <- 101
> x <- seq(ufoo[1], ufoo[2], length = nx)
> y <- seq(ufoo[3], ufoo[4], length = ny)
> xx <- outer(x, y^0)
> yy <- outer(x^0, y)
> xx <- as.vector(xx)
> yy <- as.vector(yy)
> n <- length(xx)
> newdata1 <- data.frame(BLK = factor(rep("1", n), levels = levels(out8$data$BLK)),
+     varb = factor(rep("fecund", n), levels = levels(out8$data$varb)),
+     LOGLVSfr = rep(0, n), LOGSLAfr = rep(0, n), resp = rep(1,
+         n), id = 1:n)
> newdata2 <- data.frame(BLK = factor(rep("1", n), levels = levels(out8$data$BLK)),
+     varb = factor(rep("fruit", n), levels = levels(out8$data$varb)),
+     LOGLVSfr = xx, LOGSLAfr = yy, resp = rep(1, n), id = 1:n)
> newdata <- rbind(newdata1, newdata2)
> pout <- predict(out8, newdata = newdata, varvar = varb,
+     idvar = id, root = rep(1, 2 * n))
> zz <- pout[newdata$varb == "fruit"]
> zz <- matrix(zz, nx, ny)
> contour(x, y, zz, add = TRUE)
> contour(x, y, zz, levels = c(10, 25), add = TRUE)
```

3.12 Diagnostic Plots for Paper

Here we just put Figure 3.3 and Figure 3.5 in one plot.

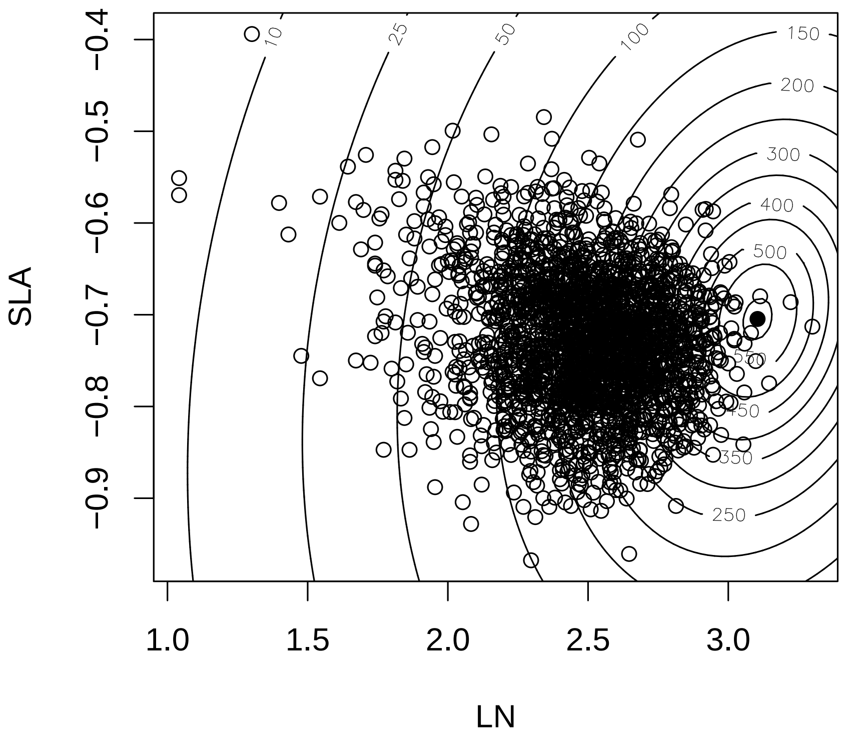

Figure 3.7: Scatterplot of `LOGLVS` versus `LOGSLA` with contours of the estimated fitness landscape. Variable names in axis labels changes to `LN` and `SLA`, respectively to agree with paper, which used these names. Solid dot is the point where the estimated fitness function achieves its maximum. Compare Figure 3.1.

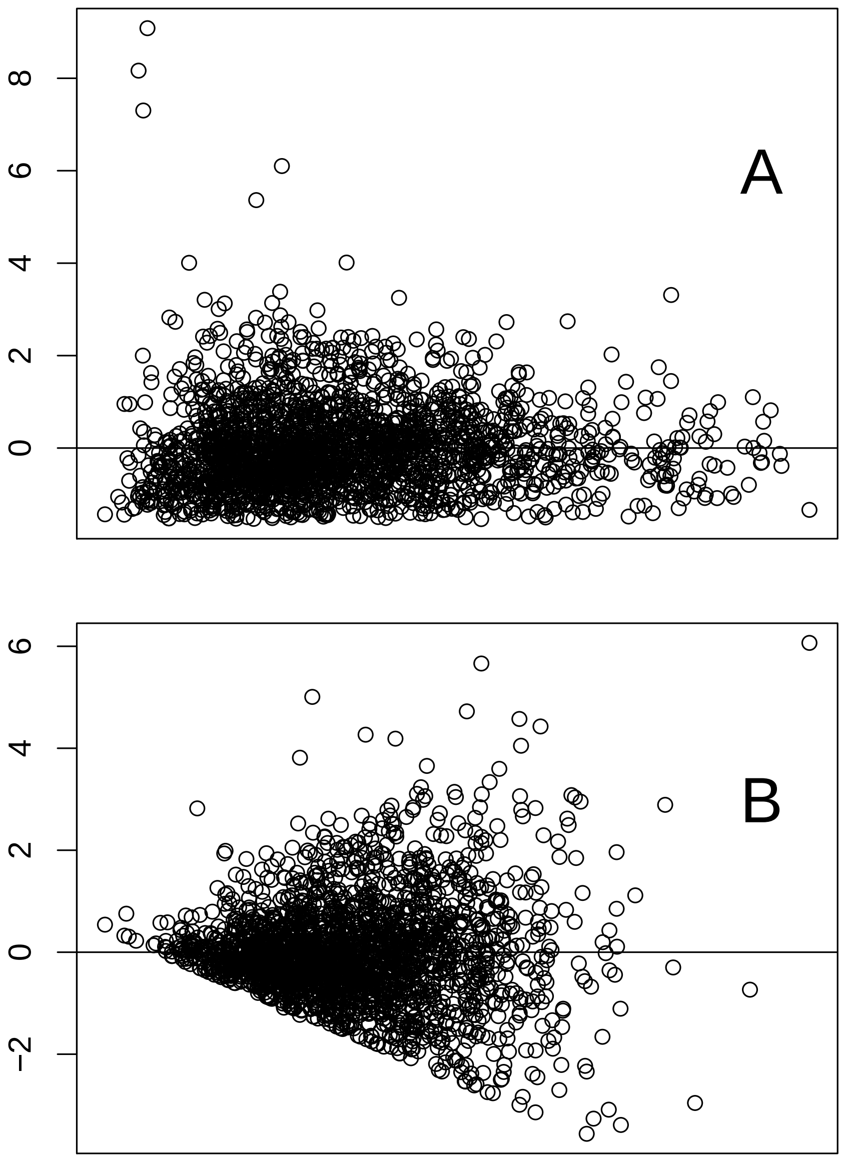

Figure 3.8: Diagnostic Plots. A: Pearson residuals for fruit count given nonzero fitness plotted against fitted values. B: standardized OLS residuals for fruit count plotted against fitted values.

Chapter 4

Lande-Arnold Analysis using Aster Models

4.1 Introduction

The analysis presented in this chapter actually was done before the analysis presented in the preceding chapter. It is an attempt to do full justice to the data. As the experiment was designed there were multiple components of fitness. For each plant that survived to that stage, fruits were counted (`fruit`) and then a random sample of fruits of size 3 was taken and the seeds in those fruits counted (texttttseed). This experimental design does not fit aster models perfectly (not the fault of the experimenters because the experiment was done before aster models were described). It would have been better if seeds were counted for all fruits or for a fraction p of fruits. Nevertheless, we do what we can, combining aster analysis and Lande-Arnold analysis. Since Lande and Arnold (1983) assume nothing about the distribution of fitness given phenotype, it is impossible to develop sampling distributions of estimates, confidence intervals, or hypothesis tests. We assume the distribution of fitness given phenotypic variables and other predictor variables is given by an aster model. Hence we can do valid statistical hypothesis tests and confidence intervals.

4.2 Data

We reanalyze a subset of the data analyzed by Etterson and Shaw (2001). Individuals of *Chamaecrista fasciculata* (common name, partridge pea) were obtained from three locations in the country and planted in three field sites. Of the complete data we only reanalyze here individuals planted in one field site (Minnesota).

These data are already in “long” format, no need to use the `reshape` function on them to do aster analysis. We will, however, need the “wide” format for Lande-Arnold analysis. So we do that, before making any changes (we will add newly defined variables) to `chamae`.

```
> library(aster)
> data(chamae)
> chamaew <- reshape(chamae, direction = "wide", timevar = "varb",
```

$$\begin{tikzcd}
 & & 1 \arrow[dl] \\
 & \texttt{fecund} \arrow[dl] \arrow[dr] & \\
\texttt{seed} & & \texttt{fruit}
\end{tikzcd}$$

Figure 4.1: Graph for *Chamaecrista* Aster Data. 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. `fecund` is Bernoulli (zero indicates no seeds, one indicates nonzero seeds). If `fecund` is zero, then so are the other variables. If `fecund` is nonzero, then `fruit` (fruit count) and `seed` (seed count) are conditionally independent, `fruit` has a two-truncated negative binomial distribution, and `seed` has a zero-truncated negative binomial distribution.

```
+     v.names = "resp", varying = list(levels(chamae$varb)))
> names(chamaew)
```

```
[1] "id"     "root"   "STG1N"  "LOGLVS" "LOGSLA" "BLK"    "fecund"
[8] "fruit"  "seed"
```

For each individual, many characteristics were measured, three of which we consider phenotypic characters (so our z is three-dimensional), and others which combine to make up an estimate of fitness. The three phenotypic characters are reproductive stage (`STG1N`), log leaf number (`LOGLVS`), and log leaf thickness (`LOGSLA`). "At the natural end of the growing season, [they] recorded total pod number and seed counts from three representative pods; from these measures, [they] estimated [fitness]" (Etterson and Shaw, 2001, further explained in their note 12).

Although aster model theory in the published version of Geyer, et al. (2007) does allow conditionally multinomial response variables, versions of the `aster` package up through 0.7-2, the current version at the time this was written, do not. Multinomial response, if we could use it, would allow us to deal individuals having seeds counted from 0, 1, 2, or 3 fruits. To avoid multinomial response, we remove individuals with seeds counted for only one or two fruits (there were only four such).

Figure 4.1 shows the graph of the aster model we use for these data. Fruit count (`fruit`) and seed count (`seed`) are dependent only in that if one is zero, then so is the other (we only model fruit count for individuals who have seeds, because fruit count for other individuals is irrelevant). Given that neither is zero (when `fecund == 1`), they are conditionally independent. Given that fruit count is nonzero, it is at least three (by our data modifications). The conditional distribution of `seed` given that it is nonzero is what is called zero-truncated negative binomial, which is negative binomial conditioned on being greater than zero. By analogy we call the conditional distribution of `fruit` given that it is nonzero, two-truncated negative binomial, which is negative binomial conditioned on being greater than two.

4.3 Aster Analysis

Then we set up the aster model framework.

```
> vars <- c("fecund", "fruit", "seed")
> pred <- c(0, 1, 1)
```

We need to choose the non-exponential-family parameters (sizes) for the negative binomial distributions, since the `aster` package only does maximum likelihood for exponential family parameters. We start with the following values, which were chosen with knowledge of the maximum likelihood estimates for these parameters, which we find in Section 4.7.

```
> load("chamae-alpha.rda")
> print(alpha.fruit)

[1] 2.48

> print(alpha.seed)

[1] 16.18

> famlist <- list(fam.bernoulli(), fam.poisson(), fam.truncated.negative.binomial(size = al
+     truncation = 0), fam.truncated.negative.binomial(size = alpha.fruit,
+     truncation = 2))
> fam <- c(1, 4, 3)
```

We can now fit our first aster model.

```
> out1 <- aster(resp ~ varb + BLK, pred, fam, varb, id,
+     root, data = chamae, famlist = famlist)
> summary(out1, show.graph = TRUE)

Call:
aster.formula(formula = resp ~ varb + BLK, pred = pred, fam = fam,
    varvar = varb, idvar = id, root = root, data = chamae, famlist = famlist)


Graphical Model:
 variable predecessor
 fecund   root
 fruit    fecund
 seed     fecund
 family
 bernoulli
 truncated.negative.binomial(size = 2.48, truncation = 2)
 truncated.negative.binomial(size = 16.18, truncation = 0)

              Estimate Std. Error  z value Pr(>|z|)
```

```
(Intercept) -1.966e+01  1.209e-01 -162.587   <2e-16 ***
varbfruit    2.064e+01  1.211e-01  170.499   <2e-16 ***
varbseed     2.019e+01  1.240e-01  162.785   <2e-16 ***
BLK2         3.928e-04  5.150e-04    0.763    0.446
BLK4         4.502e-03  4.640e-04    9.701   <2e-16 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
```

The “response” `resp` is a numeric vector containing all the response variables (`fecund`, `fruit`, and `seed`). The “predictor” `varb` is a factor with four levels distinguishing with `resp` which original response variable an element is. The predictor `BLK` has not been mentioned so far. It is block within the field where the plants were grown.

Now we add phenotypic variables.

```
> out2 <- aster(resp ~ varb + BLK + LOGLVS + LOGSLA + STG1N,
+     pred, fam, varb, id, root, data = chamae, famlist = famlist)
> summary(out2)

Call:
aster.formula(formula = resp ~ varb + BLK + LOGLVS + LOGSLA +
    STG1N, pred = pred, fam = fam, varvar = varb, idvar = id,
    root = root, data = chamae, famlist = famlist)

              Estimate Std. Error  z value Pr(>|z|)
(Intercept) -1.925e+01  1.228e-01 -156.793  < 2e-16 ***
varbfruit    2.020e+01  1.234e-01  163.759  < 2e-16 ***
varbseed     1.975e+01  1.263e-01  156.454  < 2e-16 ***
BLK2         2.483e-04  5.572e-04    0.446 0.655819
BLK4         1.994e-03  5.238e-04    3.807 0.000140 ***
LOGLVS       9.831e-03  9.381e-04   10.479  < 2e-16 ***
LOGSLA       6.099e-03  2.847e-03    2.143 0.032148 *
STG1N        5.381e-03  2.841e-04   18.941  < 2e-16 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
```

One might think we should use `varb * (LOGLVS + LOGSLA + STG1N)` but it turns out this is too many parameters and the Fisher information is ill conditioned, as shown by the need to use the `info.tol` argument.

```
> out2foo <- aster(resp ~ BLK + varb * (LOGLVS + LOGSLA +
+     STG1N), pred, fam, varb, id, root, data = chamae,
+     famlist = famlist)
> summary(out2foo, info.tol = 1e-11)

Call:
aster.formula(formula = resp ~ BLK + varb * (LOGLVS + LOGSLA +
    STG1N), pred = pred, fam = fam, varvar = varb, idvar = id,
    root = root, data = chamae, famlist = famlist)

                   Estimate Std. Error z value Pr(>|z|)
(Intercept)      -1.029e+01  1.586e+00  -6.487 8.75e-11 ***
BLK2              8.529e-04  5.787e-04   1.474 0.140559
BLK4              2.114e-03  5.574e-04   3.793 0.000149 ***
varbfruit         1.123e+01  1.587e+00   7.077 1.47e-12 ***
varbseed          1.062e+01  1.627e+00   6.530 6.59e-11 ***
LOGLVS           -4.041e+00  4.464e-01  -9.054  < 2e-16 ***
LOGSLA            1.945e+00  1.671e+00   1.164 0.244244
STG1N             1.200e+00  1.591e-01   7.541 4.68e-14 ***
varbfruit:LOGLVS  4.058e+00  4.465e-01   9.089  < 2e-16 ***
varbseed:LOGLVS   4.085e+00  4.587e-01   8.905  < 2e-16 ***
varbfruit:LOGSLA -1.942e+00  1.672e+00  -1.161 0.245511
varbseed:LOGSLA  -1.986e+00  1.710e+00  -1.161 0.245579
varbfruit:STG1N  -1.201e+00  1.593e-01  -7.540 4.70e-14 ***
varbseed:STG1N   -1.178e+00  1.637e-01  -7.197 6.16e-13 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

> anova(out2, out2foo)

Analysis of Deviance Table

Model 1: resp ~ varb + BLK + LOGLVS + LOGSLA + STG1N
Model 2: resp ~ BLK + varb * (LOGLVS + LOGSLA + STG1N)
  Model Df Model Dev Df Deviance P(>|Chi|)
1        8     85233
2       14     84432  6      802 7.12e-170
```

Despite the statistically significant improvement (based on the chi-square approximation to the log likelihood ratio, which may not be valid with such an ill-conditioned Fisher information), we do not adopt this model (`out2foo`) either.

Although we cannot afford 9 parameters (3 levels of `varb` times 3 predictor variables) for the interaction, we can afford 6, only putting the phenotype variables in at level `fruit` and `seed`. Because we are fitting an unconditional aster model, the effects of these terms are passed down to `fecund`. See the example in Geyer, et al. (2007) for discussion of this phenomenon.

```
> foo <- as.numeric(as.character(chamae$varb) == "fruit")
> chamae$LOGLVSfr <- chamae$LOGLVS * foo
> chamae$LOGSLAfr <- chamae$LOGSLA * foo
> chamae$STG1Nfr <- chamae$STG1N * foo
> foo <- as.numeric(as.character(chamae$varb) == "seed")
> chamae$LOGLVSsd <- chamae$LOGLVS * foo
> chamae$LOGSLAsd <- chamae$LOGSLA * foo
> chamae$STG1Nsd <- chamae$STG1N * foo
> out6 <- aster(resp ~ varb + BLK + LOGLVSfr + LOGSLAfr +
+     STG1Nfr + LOGLVSsd + LOGSLAsd + STG1Nsd, pred, fam,
+     varb, id, root, data = chamae, famlist = famlist)
> summary(out6)

Call:
aster.formula(formula = resp ~ varb + BLK + LOGLVSfr + LOGSLAfr +
    STG1Nfr + LOGLVSsd + LOGSLAsd + STG1Nsd, pred = pred, fam = fam,
    varvar = varb, idvar = id, root = root, data = chamae, famlist = famlist)

              Estimate Std. Error  z value Pr(>|z|)
(Intercept) -1.871e+01  1.274e-01 -146.865  < 2e-16 ***
varbfruit    1.966e+01  1.280e-01  153.615  < 2e-16 ***
varbseed     1.926e+01  1.344e-01  143.371  < 2e-16 ***
BLK2         5.049e-04  5.717e-04    0.883 0.377128
BLK4         2.105e-03  5.446e-04    3.866 0.000111 ***
LOGLVSfr     1.639e-02  1.035e-03   15.844  < 2e-16 ***
LOGSLAfr     7.287e-03  3.783e-03    1.926 0.054101 .
STG1Nfr      1.294e-04  3.146e-04    0.411 0.680958
LOGLVSsd    -7.076e-02  8.038e-03   -8.803  < 2e-16 ***
LOGSLAsd    -1.621e-03  2.904e-02   -0.056 0.955483
STG1Nsd      5.884e-02  2.978e-03   19.759  < 2e-16 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
```

When we analyzed the Minnesota-Minnesota subset alone (the subset of these data consisting of only the Minnesota population) the there was no statistically significant effect of the phenotypic predictors on seed count. In these data that effect is significant.

```
> out5 <- aster(resp ~ varb + BLK + LOGLVSfr + LOGSLAfr +
+     STG1Nfr, pred, fam, varb, id, root, data = chamae,
+     famlist = famlist)
> summary(out5)

Call:
aster.formula(formula = resp ~ varb + BLK + LOGLVSfr + LOGSLAfr +
    STG1Nfr, pred = pred, fam = fam, varvar = varb, idvar = id,
    root = root, data = chamae, famlist = famlist)

              Estimate Std. Error  z value Pr(>|z|)
(Intercept) -1.930e+01  1.224e-01 -157.764  < 2e-16 ***
varbfruit    2.025e+01  1.230e-01  164.601  < 2e-16 ***
varbseed     1.984e+01  1.254e-01  158.152  < 2e-16 ***
BLK2         1.752e-04  5.489e-04    0.319 0.749659
BLK4         1.944e-03  5.163e-04    3.765 0.000166 ***
LOGLVSfr     1.135e-02  1.004e-03   11.301  < 2e-16 ***
LOGSLAfr     6.041e-03  3.064e-03    1.972 0.048663 *
```

```
STG1Nfr      5.246e-03  2.996e-04   17.508  < 2e-16 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
```

```
> anova(out5, out6)
```

```
Analysis of Deviance Table

Model 1: resp ~ varb + BLK + LOGLVSfr + LOGSLAfr + STG1Nfr
Model 2: resp ~ varb + BLK + LOGLVSfr + LOGSLAfr + STG1Nfr + LOGLVSsd +
  Model Df Model Dev Df Deviance  P(>|Chi|)
1        8     85302
2       11     84610  3      692 1.049e-149
```

We stop our search for aster models here (using model `out6` for the rest of our analysis). Perhaps with a more diligent search we could find a slightly better fitting model, but obvious things to throw into the model (interactions) use too many parameters, so a better fitting model would have to be cleverly devised. This model fits well enough to serve as an example (see, however, the residual analyses in Section 4.6 below).

4.4 Lande-Arnold Analysis

In contrast to the aster analysis, the Lande-Arnold analysis is very simple.

```
> chamaew$fit <- chamaew$fruit * chamaew$seed
> chamaew$relfit <- chamaew$fit/mean(chamaew$fit)
> lout <- lm(relfit ~ LOGLVS + LOGSLA + STG1N, data = chamaew)
> summary(lout)
```

```
Call:
lm(formula = relfit ~ LOGLVS + LOGSLA + STG1N, data = chamaew)

Residuals:
    Min      1Q  Median      3Q     Max
-2.8589 -0.7953 -0.3592  0.2828 11.4682

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept) -3.43393    0.44764  -7.671 2.53e-14 ***
LOGLVS       1.73055    0.13352  12.961  < 2e-16 ***
LOGSLA       1.83028    0.46410   3.944 8.27e-05 ***
STG1N        0.75655    0.03507  21.571  < 2e-16 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 1.533 on 2231 degrees of freedom
Multiple R-Squared: 0.1972,        Adjusted R-squared: 0.1962
F-statistic: 182.7 on 3 and 2231 DF,  p-value: < 2.2e-16
```

The information contained in the printout of `summary(lout1)` with the exception of the `Estimate` column is unreliable because the OLS model assumptions are not satisfied, as acknowledged by Etterson and Shaw (2001) and Etterson (2004). Therefore measures of statistical significance including standard errors (`Std. Error` column), t-statistics (`t value` column), and P-values (`Pr(>|t|)` column) are erroneous.

Also the `(Intercept)` regression is of no interest (not part of β or γ).

We can also estimate $\beta(e)$ as a constant function, where e is `BLK`, our comments about that applying to OLS regression estimates as well as to (our as yet to be determined "better" estimates).

```
> loute <- lm(relfit ~ BLK + LOGLVS + LOGSLA + STG1N, data = chamaew)
> summary(loute)

Call:
lm(formula = relfit ~ BLK + LOGLVS + LOGSLA + STG1N, data = chamaew)

Residuals:
    Min      1Q  Median      3Q     Max 
-2.9551 -0.7811 -0.3143  0.2689 11.6144 

Coefficients:
            Estimate Std. Error t value Pr(>|t|)    
(Intercept) -3.33725    0.44323  -7.529 7.36e-14 ***
BLK2        -0.03046    0.07953  -0.383  0.70176    
BLK4         0.49001    0.08329   5.883 4.63e-09 ***
LOGLVS       1.51631    0.13686  11.079  < 2e-16 ***
LOGSLA       1.28264    0.46609   2.752  0.00597 ** 
STG1N        0.69622    0.03574  19.482  < 2e-16 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1 

Residual standard error: 1.517 on 2229 degrees of freedom
Multiple R-Squared: 0.2148,     Adjusted R-squared: 0.2131 
F-statistic:   122 on 5 and 2229 DF,  p-value: < 2.2e-16 

> coefficients(loute)

(Intercept)        BLK2        BLK4      LOGLVS      LOGSLA 
-3.33724548 -0.03046094  0.49000559  1.51631480  1.28263998 
      STG1N 
 0.69621813 
```

Note the large change from including e.

```
> phenonam <- c("LOGLVS", "LOGSLA", "STG1N")
> beta.hat.ols <- coefficients(lout)[phenonam]
> beta.hat.ols.e <- coefficients(loute)[phenonam]
> beta.hat.ols - beta.hat.ols.e
```

```
    LOGLVS     LOGSLA      STG1N
0.21424019 0.54763766 0.06033548
```

Although we think we should find better estimators of β and $\beta(e)$ than the OLS estimators, we work with these first.

4.5 Parametric Bootstrap

In a parametric bootstrap, the results are random. The depend on the random number generator seed and the bootstrap sample size. For sufficiently large bootstrap sample size, the dependence on the seed is small, but it still there. In order to get the same results every time, we set the seed. Anyone who does not want the same result every time (to see the randomness in the bootstrap, for example, should remove the following chunk.

```
> set.seed(42)
```

The function `raster` simulates data from an aster model. This follows the last section of the aster package vignette. We use the parameter values for the model `out6`.

```
> theta.hat <- predict(out6, model.type = "cond", parm.type = "canon")
> theta.hat <- matrix(theta.hat, nrow = nrow(out6$x), ncol = ncol(out6$x))
> root <- out6$root
> nboot <- 1000
> betastar <- matrix(NA, length(beta.hat.ols), nboot)
> betaestar <- matrix(NA, length(beta.hat.ols), nboot)
> for (i in 1:nboot) {
+     foo <- raster(theta.hat, pred, fam, root, famlist = famlist)
+     wstar <- foo[, 2] * foo[, 3]
+     wstar <- wstar/mean(wstar)
+     loutstar <- lm(wstar ~ LOGLVS + LOGSLA + STG1N, data = chamaew)
+     loutestar <- lm(wstar ~ BLK + LOGLVS + LOGSLA + STG1N,
+         data = chamaew)
+     betastar[, i] <- coefficients(loutstar)[phenonam]
+     betaestar[, i] <- coefficients(loutestar)[phenonam]
+ }
```

The matrix `betastar` contains the (parametric) bootstrap distribution of $\hat{\beta}_{\text{OLS}}$. Each row is the bootstrap distribution of one regression coefficient. Each column (a three-vector) is one bootstrap replicate of $\hat{\beta}$. The matrix `betaestar` contains the (parametric) bootstrap distribution of our analogous OLS estimator of $\beta(e)$.

Figure 4.2 (page 62) shows the histogram of the parametric bootstrap distribution of the regression coefficient for `LOGLVS`. in the estimate of β. It is given only to show that this bootstrap distribution is approximately normal, so it can be described in terms of mean and standard deviation. The distributions for the other two regression coefficients in `betastar` (not shown) are similarly approximately normal, as are those for the three regression coefficients in `betaestar`. The means and standard deviations for our OLS estimates of components of β are

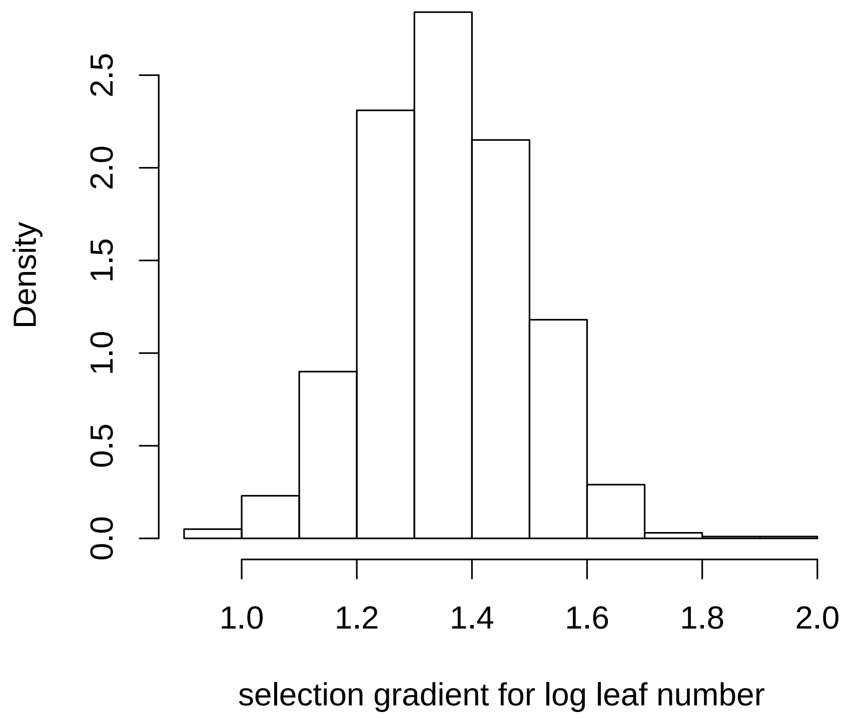

Figure 4.2: Histogram of the parametric bootstrap distribution for the selection gradient for log leaf number.

```
> meanbetastar <- apply(betastar, 1, mean)
> sdbetastar <- apply(betastar, 1, sd)
> foo <- cbind(meanbetastar, sdbetastar)
> dimnames(foo) <- list(phenonam, c("mean", "s. d."))
> print(foo)

            mean      s. d.
LOGLVS 1.3561669 0.13707633
LOGSLA 1.7897740 0.41916187
STG1N  0.8095845 0.02957681
```

For comparison the OLS estimates and nominal standard errors are

```
> foo <- summary(lout)$coefficients[, 1:2]
> print(foo)

              Estimate Std. Error
(Intercept) -3.4339326 0.44764441
LOGLVS       1.7305550 0.13352262
LOGSLA       1.8302776 0.46410116
STG1N        0.7565536 0.03507319
```

We note in passing that the means are rather different from the OLS estimates given on p. 60, the maximum absolute difference being 0.374.

The means and standard deviations for our OLS estimates of components of $\beta(e)$ are

```
> meanbetaestar <- apply(betaestar, 1, mean)
> sdbetaestar <- apply(betaestar, 1, sd)
> foo <- cbind(meanbetaestar, sdbetaestar)
> dimnames(foo) <- list(phenonam, c("mean", "s. d."))
> print(foo)

            mean      s. d.
LOGLVS 1.1877270 0.13666071
LOGSLA 1.3807413 0.42240265
STG1N  0.7662728 0.03115819
```

For comparison the OLS estimates and nominal standard errors are

```
> foo <- summary(loute)$coefficients[, 1:2]
> print(foo)

               Estimate Std. Error
(Intercept) -3.33724548 0.44323206
BLK2        -0.03046094 0.07953440
BLK4         0.49000559 0.08328757
LOGLVS       1.51631480 0.13686216
LOGSLA       1.28263998 0.46609281
STG1N        0.69621813 0.03573699
```

We note in passing that the means are rather different from the OLS estimates given on p. 60, the maximum absolute difference being 0.329.

Our bootstrap standard errors are much smaller than the OLS standard errors produced by the regression routine (which are invalid because the OLS model assumptions are invalid).

Although here we have so much data that all three regression coefficients are clearly statistically significantly different from zero, if the sample size were smaller doing the right thing might make a difference in hypothesis testing. A more important point is that our standard errors are scientifically defensible, whereas the OLS standard errors are not, since the OLS assumptions are obviously and grossly false.

4.6 Goodness of Fit

In this section we examine three issues. Is the assumed conditional independence of `fruit` and `seed` given `fecund == 1` correct? Are the assumed conditional distributions for `fruit` and `seed` given `fecund == 1` correct?

4.6.1 Independence

We tackle the easiest first. Easy in a sense because impossible. We cannot test for independence. The best we can do is a nonparametric test for lack of correlation.

```
> woof <- chamaew$fruit[chamaew$fecund == 1]
> meow <- chamaew$seed[chamaew$fecund == 1]
> cout <- cor.test(woof, meow, method = "kendall")
> print(cout)

        Kendall's rank correlation tau

data:  woof and meow
z = 4.0141, p-value = 5.967e-05
alternative hypothesis: true tau is not equal to 0
sample estimates:
       tau
0.08651135
```

The correlation (Kendall's tau) is statistically significantly different from zero, but perhaps, at 0.087 not practically significant. In any case, having no way put dependence in our aster model, we proceed as if not practically significant. Figure 4.3 (page 65) shows the scatter plot of the fitted mean value parameter (for each individual) versus the observed value for fruit count.

4.6.2 Conditional of Fruit given Nonzero Fitness

Residual analysis of generalized linear models (GLM) is tricky. (Our aster model becomes a GLM when we consider only the conditional distribution associated with one arrow.) Many different residuals have been proposed Davison and Snell (1991). We start with the simplest, so called Pearson residuals.

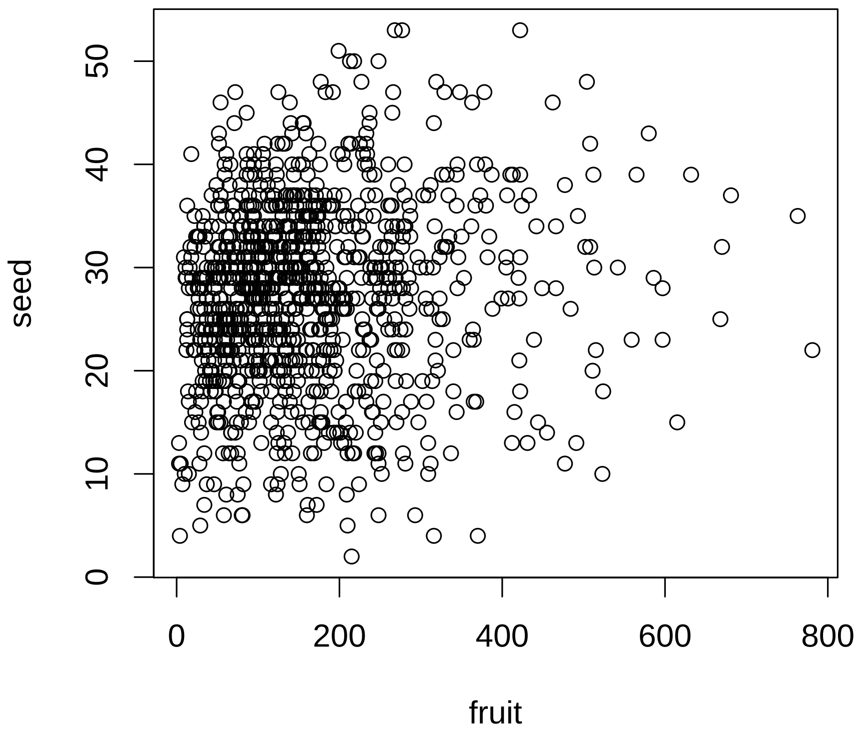

Figure 4.3: Scatter plot fruit count versus seed count conditioned on nonzero fitness.

```
> xi.hat <- predict(out6, model.type = "cond", parm.type = "mean")
> xi.hat <- matrix(xi.hat, nrow = nrow(out6$x), ncol = ncol(out6$x))
```

```
> range(woof)
```

```
[1]   3 781
```

```
> nwoof <- length(woof)
> woof.theta <- theta.hat[chamaew$fecund == 1, 2]
> woof.xi <- xi.hat[chamaew$fecund == 1, 2]
> wgrad <- double(nwoof)
> winfo <- double(nwoof)
> for (i in 1:nwoof) {
+     wgrad[i] <- famfun(famlist[[4]], deriv = 1, woof.theta[i])
+     winfo[i] <- famfun(famlist[[4]], deriv = 2, woof.theta[i])
+ }
> all.equal(woof.xi, wgrad)
```

```
[1] TRUE
```

```
> pearson <- (woof - woof.xi)/sqrt(winfo)
```

Figure 4.4 (page 67) shows the scatter plot of the Pearson residuals for fruit count plotted against the expected fruit count given that fruit count is nonzero (for each individual) for individuals with nonzero fitness only.

Figure 4.4 is not perfect. There are 1 individuals with Pearson residual greater than 6 and an additional 3 individuals with Pearson residual between 4 and 6. On the other hand, there are no individuals with Pearson residual less thatn -2. One does not expect Pearson residuals for a generalized linear model, much less an aster model, to behave as well for normal-theory linear models, but the lack of fit here is a bit worrying. The large positive "outliers" (which are not outliers in the sense of being bad data) indicate that our negative binomial model does not perfectly model these data (the negative binomial model is, however, an enormous improvement over the Poisson model).

4.6.3 Conditional of Seed given Nonzero Fitness

Now we do the analogous plot of the conditional distribution of `seed` given nonzero fitness.

```
> range(meow)
```

```
[1]  2 53
```

```
> nmeow <- length(meow)
> meow.theta <- theta.hat[chamaew$fecund == 1, 3]
> meow.xi <- xi.hat[chamaew$fecund == 1, 3]
> wgrad <- double(nmeow)
> winfo <- double(nmeow)
```

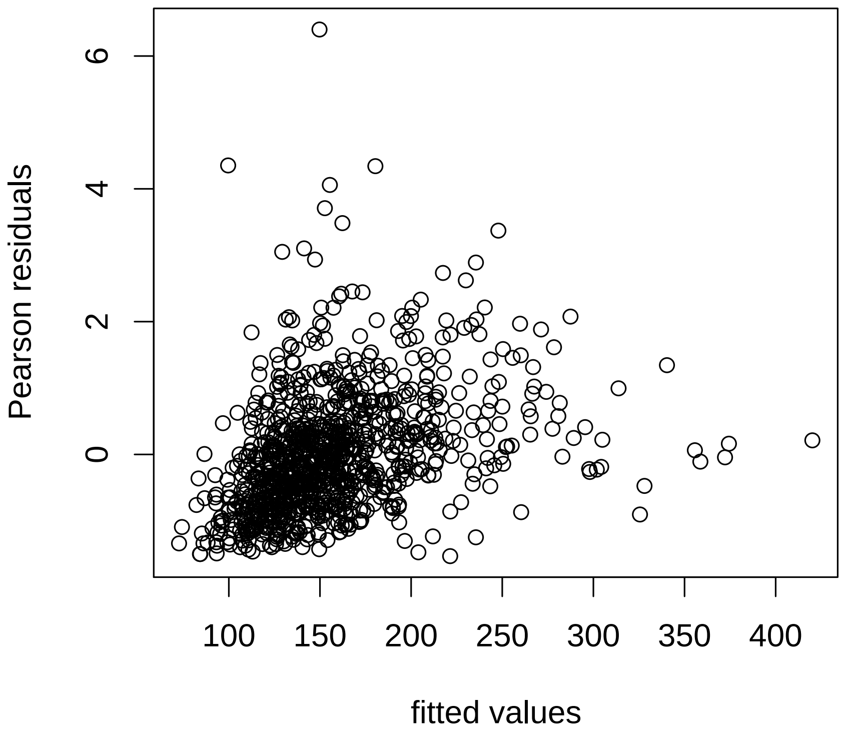

Figure 4.4: Pearson residuals for fruit count given nonzero fitness plotted against fitted values.

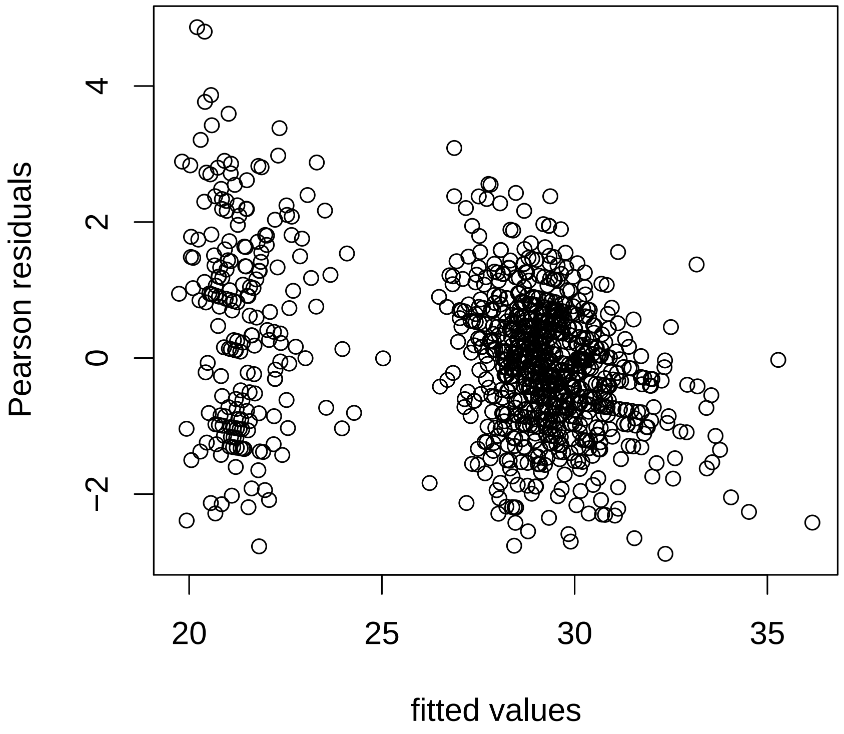

Figure 4.5: Pearson residuals for seed count given nonzero fitness plotted against fitted values.

```
> for (i in 1:nmeow) {
+     wgrad[i] <- famfun(famlist[[3]], deriv = 1, meow.theta[i])
+     winfo[i] <- famfun(famlist[[3]], deriv = 2, meow.theta[i])
+ }
> all.equal(meow.xi, wgrad)

[1] TRUE

> pearson <- (meow - meow.xi)/sqrt(winfo)
```

Figure 4.5 (page 68) shows the scatter plot of the Pearson residuals for seed count plotted against the expected seed count given that fruit count is nonzero (for each individual) for individuals with nonzero fitness only. There are no obvious problem with Figure 4.5. Certainly, it is much less troubling than Figure 4.4.

4.7 Maximum Likelihood Estimation of Size

The aster function does not calculate the correct likelihood when the size parameters are considered unknown, because it drops terms that do not involve the exponential family parameters. However, the full log likelihood is easily calculated in R.

```
> x <- out6$x
> logl <- function(alpha.fruit, alpha.seed, theta, x) {
+     x.fecund <- x[, 1]
+     theta.fecund <- theta[, 1]
+     p.fecund <- 1/(1 + exp(-theta.fecund))
+     logl.fecund <- sum(dbinom(x.fecund, 1, p.fecund,
+         log = TRUE))
+     x.fruit <- x[x.fecund == 1, 2]
+     theta.fruit <- theta[x.fecund == 1, 2]
+     p.fruit <- (-expm1(theta.fruit))
+     logl.fruit <- sum(dnbinom(x.fruit, size = alpha.fruit,
+         prob = p.fruit, log = TRUE) - pnbinom(2, size = alpha.fruit,
+         prob = p.fruit, lower.tail = FALSE, log = TRUE))
+     x.seed <- x[x.fecund == 1, 3]
+     theta.seed <- theta[x.fecund == 1, 3]
+     p.seed <- (-expm1(theta.seed))
+     logl.seed <- sum(dnbinom(x.seed, size = alpha.seed,
+         prob = p.seed, log = TRUE) - pnbinom(0, size = alpha.seed,
+         prob = p.seed, lower.tail = FALSE, log = TRUE))
+     logl.fecund + logl.fruit + logl.seed
+ }
```

We then calculate the profile likelihood for the two size parameters (alpha.fruit and alpha.seed), maximizing over the other parameters. Evaluating the profile log likelihood on a grid of points.

```
> alpha.fruit.seq <- seq(1.5, 3.5, 0.25)
> alpha.seed.seq <- seq(10, 30, 0.5)
> logl.seq <- matrix(NA, nrow = length(alpha.fruit.seq),
+     ncol = length(alpha.seed.seq))
> for (i in 1:length(alpha.fruit.seq)) {
+     for (j in 1:length(alpha.seed.seq)) {
+         famlist.seq <- famlist
+         famlist.seq[[3]] <- fam.truncated.negative.binomial(size = alpha.seed.seq[j],
+             truncation = 0)
+         famlist.seq[[4]] <- fam.truncated.negative.binomial(size = alpha.fruit.seq[i],
+             truncation = 2)
+         out6.seq <- aster(out6$formula, pred, fam, varb,
+             id, root, data = chamae, famlist = famlist.seq,
+             parm = out6$coefficients)
+         theta.seq <- predict(out6.seq, model.type = "cond",
+             parm.type = "canon")
+         dim(theta.seq) <- dim(x)
+         logl.seq[i, j] <- logl(alpha.fruit.seq[i], alpha.seed.seq[j],
+             theta.seq, x)
+     }
+ }
> alpha.fruit.interp <- seq(min(alpha.fruit.seq), max(alpha.fruit.seq),
+     0.01)
> alpha.seed.interp <- seq(min(alpha.seed.seq), max(alpha.seed.seq),
+     0.01)
> logl.foo <- matrix(NA, nrow = length(alpha.fruit.interp),
+     ncol = length(alpha.seed.seq))
> for (i in 1:length(alpha.seed.seq)) logl.foo[, i] <- spline(alpha.fruit.seq,
+     logl.seq[, i], n = length(alpha.fruit.interp))$y
> logl.bar <- matrix(NA, nrow = length(alpha.fruit.interp),
+     ncol = length(alpha.seed.interp))
> for (i in 1:length(alpha.fruit.interp)) logl.bar[i, ] <- spline(alpha.seed.seq,
+     logl.foo[i, ], n = length(alpha.seed.interp))$y
> imax.fruit <- row(logl.bar)[logl.bar == max(logl.bar)]
> imax.seed <- col(logl.bar)[logl.bar == max(logl.bar)]
> alpha.fruit <- alpha.fruit.interp[imax.fruit]
> alpha.seed <- alpha.seed.interp[imax.seed]
> save(alpha.fruit, alpha.seed, file = "chamae-alpha.rda",
+     ascii = TRUE)
```

At the end of this chunk we save the maximum likelihood estimates in a file which is read in at the beginning of this document.

Figure 4.6 (page 71) shows the profile log likelihood for the size parameters.

4.8 OLS Diagnostic Plots

Although unnecessary because we know the assumptions justifying OLS are badly violated, here are some diagnostic plots for the OLS regression.

Figure 4.7 (page 72) shows the plot of residuals versus fitted values made by the R statement

```
> plot(loute, which = 1)
```

Figure 4.8 (page 73) shows the Normal Q-Q (quantile-quantile) plot made by the R statement

```
> plot(loute, which = 2)
```

Clearly the errors are highly non-normal.

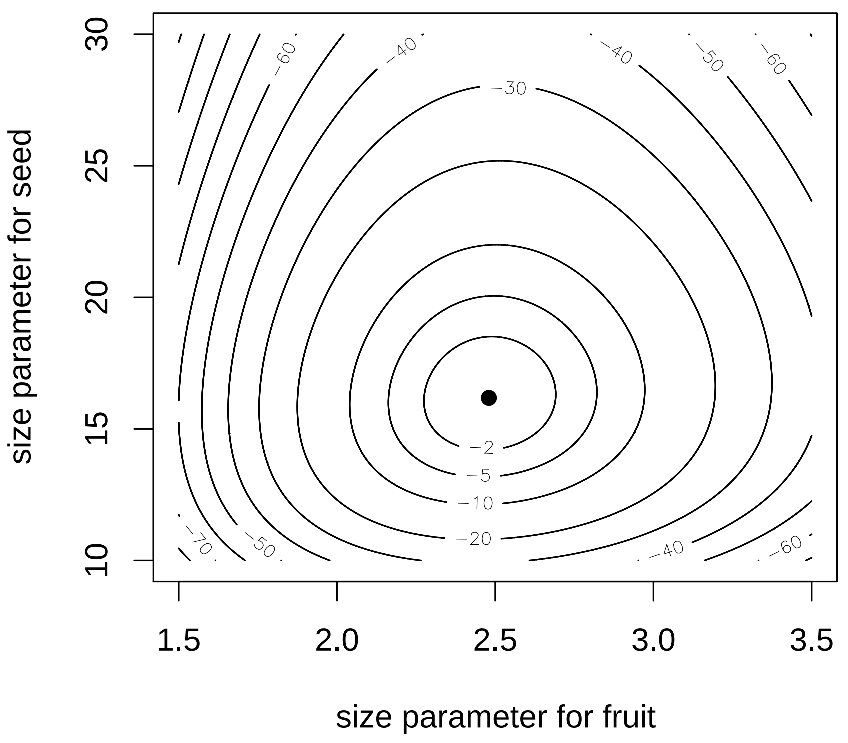

Figure 4.6: Profile log likelihood for size parameters for the negative binomial distributions of fruit and seed. Solid dot is maximum likelihood estimate.

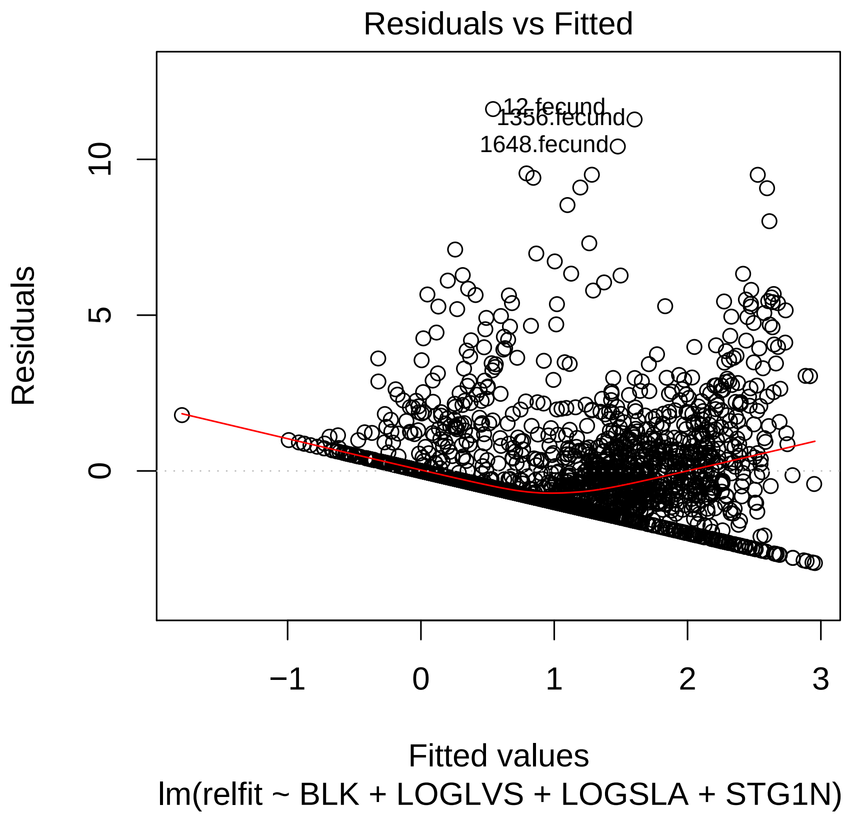

Figure 4.7: Residuals versus Fitted plot for OLS fit with blocks.

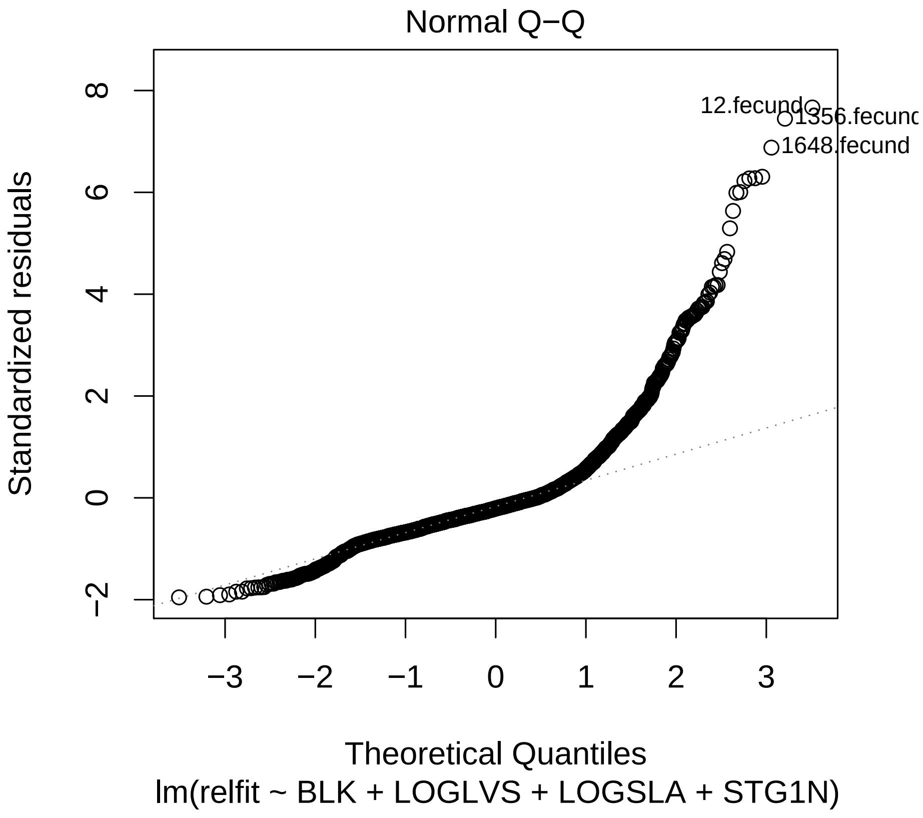

Figure 4.8: Normal Q-Q plot for OLS fit with blocks.

Chapter 5

Aster Analysis of Growth Rate

5.1 Introduction

5.1.1 Data

Lenski and Service (1982) give data (their Table 2) that is ideal for aster model analysis (Geyer, et al., 2007). These data are in a dataset `aphid` in version 0.7-2 or later of the `aster` contributed package for R.

```
> library(aster)
> data(aphid)
> names(aphid)

[1] "root" "varb" "resp" "id"
```

These data are already in "long" format, no need to use the `reshape` function on them to do aster analysis. The "original" variables, those describe in Lenski and Service (1982), are all in the variable `resp`. Which components of `resp` correspond to which "original" variable is indicated by the variable `varb`, which has levels

```
> levels(aphid$varb)

 [1] "B2"  "B3"  "B4"  "B5"  "B6"  "B7"  "B8"  "B9"  "S1"  "S10"
[11] "S11" "S12" "S13" "S2"  "S3"  "S4"  "S5"  "S6"  "S7"  "S8" 
[21] "S9" 
```

Which components of `resp` correspond to which "original" individual is indicated by the variable `id`, which has unique values

```
> sort(unique(aphid$id))

 [1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18
```

so we have the 18 individuals recorded in the data of Lenski and Service (1982).

The "original" variables labeled Sx, where x is a number between 1 and 13, are survival indicators (one for alive, zero for dead), and `root` is all ones (every individual was alive at the start of data collection). Our `root` corresponds to the `S0` of Lenski and Service (1982). The "original" variables labeled Bx where x is a number between 2 and 9 are the number of offspring born to that individual (all individuals are female aphids) in that time period.

$$\begin{tikzcd}
1 \arrow[r] & \mathtt{S1} \arrow[r] & \mathtt{S2} \arrow[r] \arrow[d] & \mathtt{S3} \arrow[r] \arrow[d] & \mathtt{S4} \arrow[r] \arrow[d] & \mathtt{S5} \arrow[r] \arrow[d] & \mathtt{S6} \arrow[r] \arrow[d] & \mathtt{S7} \arrow[r] \arrow[d] & \mathtt{S8} \arrow[r] \arrow[d] & \mathtt{S9} \arrow[r] \arrow[d] & \mathtt{S10} \arrow[r] & \mathtt{S11} \arrow[r] & \mathtt{S12} \arrow[r] & \mathtt{S13} \\
 & & \mathtt{B2} & \mathtt{B3} & \mathtt{B4} & \mathtt{B5} & \mathtt{B6} & \mathtt{B7} & \mathtt{B8} & \mathtt{B9} & & & &
\end{tikzcd}$$

Figure 5.1: Graph for *Uroleucon rudbeckiae* data. Arrows go from one life history component to another indicating conditional dependence in the aster model. Nodes are labeled by their associated variables. Root nodes are associated with the constant variable 1, indicating presence of individuals at the outset. If any parent variable is zero, then the child variable is also zero. Child variables are conditionally independent given the parent variable. If a parent variable is nonzero, then the conditional distribution of the child variable is as follows. Si is (conditionally) Bernoulli (zero indicates mortality, one indicates survival) and Bi is (conditionally) zero-truncated Poisson.

5.1.2 Aster Model

We use an aster model for these data described as follows.

- Variable `root` is the predecessor of the variable `S1`.
- Variable `S1` is Bernoulli.
- Variable Sx is the predecessor of the variable Bx.
- Variable Bx is Poisson given the variable Sx.
- Variable S$(x-1)$ is the predecessor of the variable Sx.
- Variable Sx is Bernoulli given the variable S$(x-1)$.

The graph for this model is shown in Figure 5.1.

The data of Lenski and Service (1982) included variables `B1` though `B13`. We have deleted some of them from the data. If we had included them and fit a saturated aster model, then we have in addition to the structural zeroes in the data (dead individuals stay dead and cannot reproduce) non-structural zeroes (zeroes in the data that are not forced by the model structure). Because we intend to use a saturated model for fecundity, for the purposes of this analysis only we deleted the Bx variables that were all zero. We do not recommend this in general nor do we claim this is the only way to analyze these data. However the question of what to do with these non-structural zeroes is difficult, an open research question in statistics, and we do not wish to complicate our example with that. If we were to use a non-saturated model, this would avoid the non-structural zeroes problem, but would require us to model fecundity as a function of time. We do not wish to do that either.

In contrast, we will attempt to model survivorship as a function of time, not so much because that is easier (though it may be), but just to illustrate both approaches. Thus we do not need to drop any Sx variables.

We now construct the aster model graphical structure as follows.

```
> varb <- unique(as.character(aphid$varb))
> varb.letter <- substr(varb, 1, 1)
> varb.number <- as.numeric(substr(varb, 2, 10))
> varb.letter

 [1] "S" "S" "B" "S" "B" "S" "B" "S" "B" "S" "B" "S" "B" "S" "B"
[16] "S" "B" "S" "S" "S" "S"

> varb.number

 [1]  1  2  2  3  3  4  4  5  5  6  6  7  7  8  8  9  9 10 11 12
[21] 13

> pred <- rep(0, length(varb))
> indx <- seq(along = varb)
> from <- indx[varb.letter == "B"]
> tovar <- paste("S", varb.number[from], sep = "")
> data.frame(from = varb[from], to = tovar)

  from to
1   B2 S2
2   B3 S3
3   B4 S4
4   B5 S5
5   B6 S6
6   B7 S7
7   B8 S8
8   B9 S9

> to <- match(tovar, varb)
> pred[from] <- to
> pred

 [1]  0  0  2  0  4  0  6  0  8  0 10  0 12  0 14  0 16  0  0  0
[21]  0

> from <- indx[varb.letter == "S"]
> tovar <- paste("S", varb.number[from] - 1, sep = "")
> data.frame(from = varb[from], to = tovar)

   from  to
1    S1  S0
2    S2  S1
3    S3  S2
4    S4  S3
5    S5  S4
6    S6  S5
```

```
7    S7  S6
8    S8  S7
9    S9  S8
10  S10  S9
11  S11 S10
12  S12 S11
13  S13 S12
```

```
> to <- match(tovar, varb)
> pred[from] <- to
> pred
```

```
 [1] NA  1  2  2  4  4  6  6  8  8 10 10 12 12 14 14 16 16 18 19
[21] 20
```

```
> data.frame(from = varb, to = varb[pred])
```

```
   from   to
1    S1 <NA>
2    S2   S1
3    B2   S2
4    S3   S2
5    B3   S3
6    S4   S3
7    B4   S4
8    S5   S4
9    B5   S5
10   S6   S5
11   B6   S6
12   S7   S6
13   B7   S7
14   S8   S7
15   B8   S8
16   S9   S8
17   B9   S9
18  S10   S9
19  S11  S10
20  S12  S11
21  S13  S12
```

```
> pred[is.na(pred)] <- 0
```

And the aster model family structure.

```
> fam <- rep(1, length(varb))
> fam[varb.letter == "B"] <- 2
```

5.2 Model Fitting

Unlike the examples discussed in Geyer, et al. (2007), these data require *conditional* aster models. We are interested in modeling survivorship and fecundity as instantaneous functions of time (or as close to that as we can get with discrete time periods). The use of unconditional aster models to address lifetime fitness, so prominent in Geyer, et al. (2007), is missing in this application.

5.2.1 Model One

We start with constant mortality rate.

```
> barb <- as.factor(sub("S[0-9]*", "S", as.character(aphid$varb)))
> aphid <- data.frame(aphid, barb = barb)
> out1 <- aster(resp ~ barb, pred, fam, varb, id, root = root,
+     data = aphid, type = "conditional")
> summary(out1, show.graph = TRUE)
```

```
Call:
aster.formula(formula = resp ~ barb, pred = pred, fam = fam,
    varvar = varb, idvar = id, root = root, data = aphid, type = "conditional")


Graphical Model:
 variable predecessor family
 S1       root        bernoulli
 S2       S1          bernoulli
 B2       S2          poisson
 S3       S2          bernoulli
 B3       S3          poisson
 S4       S3          bernoulli
 B4       S4          poisson
 S5       S4          bernoulli
 B5       S5          poisson
 S6       S5          bernoulli
 B6       S6          poisson
 S7       S6          bernoulli
 B7       S7          poisson
 S8       S7          bernoulli
 B8       S8          poisson
 S9       S8          bernoulli
 B9       S9          poisson
 S10      S9          bernoulli
 S11      S10         bernoulli
 S12      S11         bernoulli
 S13      S12         bernoulli
```

```
            Estimate Std. Error z value Pr(>|z|)
(Intercept)   0.2513     0.2398   1.048 0.294731
barbB3        1.2368     0.2771   4.463 8.09e-06 ***
barbB4        1.0014     0.2927   3.422 0.000622 ***
barbB5        0.8938     0.3061   2.920 0.003497 **
barbB6        0.4026     0.3545   1.136 0.256134
barbB7        0.1234     0.4019   0.307 0.758855
barbB8       -1.0622     0.6710  -1.583 0.113410
barbB9       -2.0431     1.1251  -1.816 0.069393 .
barbS         1.6205     0.3653   4.436 9.16e-06 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
```

5.2.2 Model Two

We add a term linear in time for survival (not for fecundity).

```
> tim <- as.numeric(substr(as.character(aphid$varb),
+     2, 10))
> tim[grep("B", as.character(aphid$varb))] <- 0
> aphid <- data.frame(aphid, tim = tim)
> out2 <- aster(resp ~ barb + tim, pred, fam, varb,
+     id, root = root, data = aphid, type = "conditional")
> summary(out2)

Call:
aster.formula(formula = resp ~ barb + tim, pred = pred, fam = fam,
    varvar = varb, idvar = id, root = root, data = aphid, type = "conditional")

            Estimate Std. Error z value Pr(>|z|)
(Intercept)  0.25131    0.21637   1.162 0.245435
barbB3       1.23676    0.24749   4.997 5.82e-07 ***
barbB4       1.00145    0.25806   3.881 0.000104 ***
barbB5       0.89382    0.26698   3.348 0.000814 ***
barbB6       0.40261    0.30442   1.323 0.185984
barbB7       0.12338    0.34542   0.357 0.720954
barbB8      -1.06224    0.58820  -1.806 0.070931 .
barbB9      -2.04307    1.05913  -1.929 0.053729 .
barbS        3.38538    0.69597   4.864 1.15e-06 ***
tim         -0.28661    0.08784  -3.263 0.001103 **
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
```

5.2.3 Model Three

We add a term quadratic in time for survival (not for fecundity).

```
> out3 <- aster(resp ~ barb + tim + I(tim^2), pred,
+     fam, varb, id, root = root, data = aphid, type = "conditional")
> summary(out3)

Call:
aster.formula(formula = resp ~ barb + tim + I(tim^2), pred = pred,
    fam = fam, varvar = varb, idvar = id, root = root, data = aphid,
    type = "conditional")

            Estimate Std. Error z value Pr(>|z|)
(Intercept)  0.25131    0.22808   1.102 0.270520
barbB3       1.23676    0.26104   4.738 2.16e-06 ***
barbB4       1.00145    0.27148   3.689 0.000225 ***
barbB5       0.89382    0.27885   3.205 0.001349 **
barbB6       0.40261    0.31205   1.290 0.196976
barbB7       0.12338    0.34392   0.359 0.719784
barbB8      -1.06224    0.55076  -1.929 0.053768 .
barbB9      -2.04307    0.95105  -2.148 0.031695 *
barbS        1.49361    0.92541   1.614 0.106529
tim          0.49977    0.34729   1.439 0.150133
I(tim^2)    -0.06100    0.02772  -2.200 0.027789 *
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

> anova(out1, out2, out3)

Analysis of Deviance Table

Model 1: resp ~ barb
Model 2: resp ~ barb + tim
Model 3: resp ~ barb + tim + I(tim^2)
  Model Df Model Dev Df Deviance P(>|Chi|)
1        9    92.295
2       10    80.749  1   11.546     0.001
3       11    75.119  1    5.630     0.018
```

Everything we have put in seems statistically significant, but we stop here, since modeling is not the main point of the example.

5.3 Population Growth Rate

All of this is nice, but it does not directly address the question of interest to Lenski and Service (1982). They are interested in the population growth rate ϕ, which we can get a point estimate for using our methods as follows.

5.3.1 Prediction I

First we form “new data” for prediction that corresponds to just one individual in the old data.

```
> renewdata <- aphid[aphid$id == 1, ]
> class(renewdata)

[1] "data.frame"

> names(renewdata)

[1] "root" "varb" "resp" "id"   "barb" "tim"

> dim(renewdata)

[1] 21  6

> nind <- 1
> nnode <- length(varb)
> prednames <- grep("B", varb, value = TRUE)
> prednames

[1] "B2" "B3" "B4" "B5" "B6" "B7" "B8" "B9"

> predno <- as.numeric(substr(prednames, 2, 10))
> predno

[1] 2 3 4 5 6 7 8 9

> npred <- length(prednames)
> amat <- array(0, c(nind, nnode, npred))
> identical(varb, as.character(renewdata$varb))

[1] TRUE

> for (i in 1:npred) amat[1, varb == prednames[i],
+     i] <- 1
> amat[1, , ]

      [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8]
 [1,]    0    0    0    0    0    0    0    0
 [2,]    0    0    0    0    0    0    0    0
 [3,]    1    0    0    0    0    0    0    0
 [4,]    0    0    0    0    0    0    0    0
 [5,]    0    1    0    0    0    0    0    0
 [6,]    0    0    0    0    0    0    0    0
 [7,]    0    0    1    0    0    0    0    0
 [8,]    0    0    0    0    0    0    0    0
 [9,]    0    0    0    1    0    0    0    0
[10,]    0    0    0    0    0    0    0    0
[11,]    0    0    0    0    1    0    0    0
[12,]    0    0    0    0    0    0    0    0
[13,]    0    0    0    0    0    1    0    0
[14,]    0    0    0    0    0    0    0    0
[15,]    0    0    0    0    0    0    1    0
[16,]    0    0    0    0    0    0    0    0
[17,]    0    0    0    0    0    0    0    1
[18,]    0    0    0    0    0    0    0    0
[19,]    0    0    0    0    0    0    0    0
[20,]    0    0    0    0    0    0    0    0
[21,]    0    0    0    0    0    0    0    0
```

```
> class(out3$modmat)
```

```
[1] "array"
```

```
> dim(out3$modmat)
```

```
[1] 18 21 11
```

```
> class(out3)
```

```
[1] "aster.formula" "aster"
```

```
> tout3 <- predict(out3, varvar = varb, idvar = id,
+     root = root, newdata = renewdata, amat = amat)
> names(tout3) <- prednames
> tout3
```

```
        B2         B3         B4         B5         B6 
1.06794655 3.44579803 2.56240419 2.15856319 1.22491862 
        B7         B8         B9 
0.83848662 0.22106259 0.06516996 
```

These are the unconditional expectations of the Bx variables indicated by the names. We claim these correspond to $\sigma_x \beta_x$ (this product) in Lenski and Service (1982).

5.3.2 Point Estimation

To simplify notation we write $\mu_x = \sigma_x \beta_x$ so equation (1) in Lenski and Service (1982) becomes

$$1 = \sum_{x=0}^{\infty} \phi^{-(x+1)} \mu_x. \tag{5.1}$$

Of course, here the sum is finite, since we only have (nonzero) μ_x for x in the R variable `predno`, ranging from 2 to 9.

The corresponding point estimate is found as follows.

```
> foo <- function(phi) sum(phi^(-(predno + 1)) * tout3) -
+     1
> uout <- uniroot(foo, lower = 1, upper = 2)
> uout

$root
[1] 1.67741

$f.root
[1] 2.956994e-06

$iter
[1] 8

$estim.prec
[1] 6.103516e-05

> phihat <- uout$root
```

Our point estimate is 1.67741. Note this is different from the estimates presented in Table 3 in Lenski and Service (1982), although not much different from any of their bias-corrected estimators (F', F^*, and F'').

5.3.3 The Delta Method

The `aster` package does not automatically do standard errors for nonlinear functions such as the function defined by

```
> dophi <- function(mu) {
+     foo <- function(phi) sum(phi^(-(predno + 1)) *
+         mu) - 1
+     uout <- uniroot(foo, lower = 1, upper = 2)
+     return(uout$root)
+ }
```

In order to apply the delta method we must find the gradient (vector of partial derivatives $\partial\phi/\partial\mu_x$) of the function defined implicitly by (5.1) and explicitly by the R function `dophi`.

Differentiating (5.1) with respect to μ_y we get

$$0 = \sum_{x=0}^{\infty} -(x+1)\phi^{-(x+2)} \frac{\partial \phi}{\partial \mu_y} \mu_x + \phi^{-(y+1)}$$

which, since the $\partial\phi/\partial\mu_y$ does not contain x and can be pulled outside the sum, can be solved for $\partial\phi/\partial\mu_y$ giving

$$\frac{\partial \phi}{\partial \mu_y} = \left(\sum_{x=0}^{\infty} (x+1)\phi^{-(x+2)+(y+1)} \mu_x \right)^{-1} \tag{5.2}$$

The partial derivatives are functions of the vector $\boldsymbol{\mu}$ having components μ_y. So we should write $\partial \phi(\boldsymbol{\mu}) / \partial \mu_y$ to denote evaluation of the partial derivatives at the point $\boldsymbol{\mu}$.

Then the delta method says the estimator $\phi(\hat{\boldsymbol{\mu}})$ has the same asymptotic variance as its linearization

$$\phi_{\text{lin}}(\hat{\boldsymbol{\mu}}) = \phi(\boldsymbol{\mu}) + \sum_{x=0}^{\infty} \frac{\partial \phi(\boldsymbol{\mu})}{\partial \mu_x} (\hat{\mu}_x - \mu_x) \tag{5.3}$$

(Taylor series about $\boldsymbol{\mu}$ with only zero-order and first-order terms).

Since ϕ_{lin} is a linear function of $\hat{\boldsymbol{\mu}}$, we can calculate its asymptotic standard deviation using the `aster` package.

```
> aphimat <- array(0, c(1, nnode, 1))
> for (i in 1:npred) aphimat[1, varb == prednames[i],
+     1] <- 1/sum((predno + 1) * phihat^(-(predno +
+     2) + (predno[i] + 1)) * tout3)
> aphimat[1, , 1]

 [1] 0.000000000 0.000000000 0.082172654 0.000000000 0.048987821
 [6] 0.000000000 0.029204444 0.000000000 0.017410440 0.000000000
[11] 0.010379360 0.000000000 0.006187730 0.000000000 0.003688860
[16] 0.000000000 0.002199141 0.000000000 0.000000000 0.000000000
[21] 0.000000000
```

```
> tout3sd <- predict(out3, varvar = varb, idvar = id,
+     root = root, newdata = renewdata, amat = aphimat,
+     se.fit = TRUE)
> tout3sd$se.fit

[1] 0.05608055
```

This is much smaller than the standard errors derived from Table 3 in Lenski and Service (1982), but this is only to be expected. Parametric estimators are generally more accurate than nonparametric estimators (when the parametric model is correct).

Note that from the last call to `predict.aster` we used only the standard error, not the point estimate. The estimate `phihat` had already been derived. Moreover, since `aster.predict` does not allow a constant term in the linear functions it estimates, we cannot make it estimate (5.3). This does not matter, since we have gotten the desired point estimate $\phi(\hat{\boldsymbol{\mu}}) = 1.677$ from our earlier calculation. The second calculation is only to get the standard errors of both $\phi(\hat{\boldsymbol{\mu}})$ and $\phi_{\text{lin}}(\hat{\boldsymbol{\mu}})$, which are the same 0.056.

5.3.4 Check of the Delta Method

We can derive the partial derivatives used in the delta method by finite difference approximation. Consider the first

```
> epsilon <- 1e-05
> dophi <- function(phi) {
```

```
+     foo <- function(phi) sum(phi^(-(predno + 1)) *
+         tout3) - 1
+     uout <- uniroot(foo, lower = 1, upper = 2, tol = 1e-08)
+     return(uout$root)
+ }
> dophi <- function(mu) {
+     foo <- function(phi) sum(phi^(-(predno + 1)) *
+         mu) - 1
+     uout <- uniroot(foo, lower = 1, upper = 2, tol = 1e-08)
+     return(uout$root)
+ }
> mueps <- tout3
> mueps[1] <- mueps[1] + epsilon
> (dophi(mueps) - dophi(tout3))/epsilon

[1] 0.0821728

> aphimat[1, 3, 1]

[1] 0.08217265
```

Pretty close.

5.4 Discussion

The point of this example is not that our methods are better than those of Lenski and Service (1982). Ours being parametric and theirs being nonparametric, ours are better when the parametric model we use is correct (or nearly so), and theirs are better otherwise, assuming the sample size is large enough for the usual asymptotics of maximum likelihood to work (for our methods) or for the jackknife to work (for theirs).

Our point is rather different: aster models can be made to do this other kind of life history analysis (LHA), which is rather different from the kind done in the example in Geyer, et al. (2007). Because both kinds are done in the same framework, this means it is possible to do analyses which have some aspects of both kinds of LHA. It stands to reason that many other kinds of LHA can be placed in the aster framework. Our story is about *unification*, not about one particular analysis being better than another for one particular data set.

An unrelated lesson is that what Geyer, et al. (2007) call a "limitation" of the `aster` package, that its `predict.aster` function handles only linear functions of the various parameterizations known to it ($\boldsymbol{\beta}$, $\boldsymbol{\theta}$, $\boldsymbol{\varphi}$, $\boldsymbol{\tau}$, and $\boldsymbol{\xi}$), is a limitation only in terms of ease of use. The package can be made to handle nonlinear functions, if one is willing and able to do the delta method partially by hand (as we did here).

Bibliography

Buckheit, J. B. and Donoho, D. L. (1995). WaveLab and reproducible research. `http://www-stat.stanford.edu/~donoho/Reports/1995/wavelab.pdf`

Davison, A. C., and Snell, E. J. (1991). Residuals and diagnostics. In *Statistical Theory and Modelling: In honour of Sir David Cox, FRS.* D. V. Hinkley, N. Reid, E. J. Snell (eds.) Chapman & Hall.

Etterson, J. R. (2004) Evolutionary potential of *Chamaecrista fasciculata* in relation to climate change. I. Clinal patterns of selection along an environmental gradient in the great plains. *Evolution*, **58**, 1446–1458.

Etterson, J. R., and Shaw, R. G. (2001). Constraint to adaptive evolution in response to global warming. *Science*, **294**, 151–154.

Gentleman, R. and Temple Lang, D. (2004). Statistical analyses and reproducible research. Bioconductor Project Working Papers. Working Paper 2. `http://www.bepress.com/bioconductor/paper2`

Geyer, C. J., Wagenius, S. and Shaw, R. G. (2007). Aster models for life history analysis. *Biometrika*, in press.

Lande, R. and Arnold, S. J. (1983). The measurement of selection on correlated characters. *Evolution*, **37**, 1210–1226.

Leisch, F. (2002a). Sweave, part I: Mixing R and LATEX. *R News*, **2**, 28–31. `http://cran.r-project.org/doc/Rnews`.

Leisch, F. (2002b). Sweave: Dynamic generation of statistical reports using literate data analysis. In Wolfgang Härdle and Bernd Rönz, editors, *Compstat 2002 — Proceedings in Computational Statistics*, pp. 575–580. Physica Verlag, Heidelberg.

Lenski, R. E. and Service, P. M. (1982). The statistical analysis of population growth rates calculated from schedules of survivorship and fecunidity. *Ecology*, **63**, 655–662.

McCullagh, P., and Nelder, J. A. (1989). *Generalized Linear Models*, 2nd ed. Chapman & Hall.

Mitchell-Olds, T., and Shaw, R. G. (1987). Regression analysis of natural selection: Statistical inference and biological interpretation. *Evolution*, **41**, 1149–1161.

R Development Core Team (2006). R: A language and environment for statistical computing. R Foundation for Statistical Computing, Vienna, Austria. `http://www.R-project.org`.

Shaw, R. G., Geyer, C. J., Wagenius, S., Hangelbroek, H. H., and Etterson, J. R. (submitted). Unifying life history analysis for inference of fitness and population growth. `http://www.stat.umn.edu/geyer/aster/`

Stanton, M. L. and Thiede, D. A. (2005). Statistical convenience vs biological insight: consequences of data transformation for the analysis of fitness variation in heterogeneous environments *New Phytologist*, **166**, 319–338.

Stein, C. M. (1981). Estimation of the mean of a multivariate normal distribution. Annals of Statistics, **9**, 1135–1151.