Three-year fitness of *Panicum virgatum* infected with switchgrass mosaic virus

Michael P. Ryskamp and Charles J. Geyer

January 8, 2023

1 R

- The version of R used to make this document is 4.2.1.
- The version of R package `rmarkdown` used to make this document is 2.19.
- The version of R package `knitr` used to make this document is 1.41.
- The version of R package `aster` used to make this document is 1.1.2.
- The version of R package `trust` used to make this document is 0.1.8.
- The version of R package `numDeriv` used to make this document is 2016.8.1.1.
- The version of R package `freshr` used to make this document is 1.0.2.

Ensure a clean R global environment.

```
freshr::freshr()
```

Load R packages `aster` and `numDeriv`

```
library("aster")
library("numDeriv")
```

Set global option.

```
# don't need this with R-4.0.0, it's the default there and forevermore
# but it doesn't hurt and defends against users who haven't upgraded R
options(stringsAsFactors = FALSE)
```

2 Data

```
redata <- read.csv("redata_fin.csv")
# "symp.extent" is the proportion of diseased tillers. Taken initially,
# in early summer 2017, and, for 2017 and 2018,
# taken at the end of season around the time
# we collected fitness metrics (panicle counts, lengths)


sapply(redata, class)
```

```
##     PlantID        Year symp.extent        varb        resp          id
## "character"   "integer"   "numeric" "character"   "integer"   "integer"
##        root
```

```
##   "integer"
```

```r
unique(redata$varb)
```

```
##  [1] "late17.tillers"           "late17.panicles"
##  [3] "late17.pans.sampled"      "late17.floret.count.total"
##  [5] "late18.tillers"           "late18.panicles"
##  [7] "late18.pans.sampled"      "late18.floret.count.total"
##  [9] "late19.tillers"           "late19.panicles"
## [11] "late19.pans.sampled"      "late19.floret.count.total"
```

```r
str(redata)
```

```
## 'data.frame':    120 obs. of  7 variables:
##  $ PlantID    : chr  "A" "B" "C" "D" ...
##  $ Year       : int  2017 2017 2017 2017 2017 2017 2017 2017 2017 2017 ...
##  $ symp.extent: num  0.42 0.42 0.39 0.68 0.19 0.05 0.14 0.04 0.22 0.22 ...
##  $ varb       : chr  "late17.tillers" "late17.tillers" "late17.tillers" "late17.tillers" ...
##  $ resp       : int  56 45 77 79 148 114 172 157 90 219 ...
##  $ id         : int  1 2 3 4 5 6 7 8 9 10 ...
##  $ root       : int  1 1 1 1 1 1 1 1 1 1 ...
```

```r
redata$PlantID <- as.factor(redata$PlantID)
redata$Year <- as.factor(redata$Year)
```

3 Graph

We use the following aster graph for one individual.

$$
\begin{tikzcd}
 & y_1 \arrow[r, "\text{Ber}"] & y_2 \arrow[r, "\text{Samp}"] & y_3 \arrow[r, "\text{Poi}"] & y_4 \\
1 \arrow[ur, "\text{Poi}"] \arrow[r, "\text{Poi}"] \arrow[dr, "\text{Poi}"] & y_5 \arrow[r, "\text{Ber}"] & y_6 \arrow[r, "\text{Samp}"] & y_7 \arrow[r, "\text{Poi}"] & y_8 \\
 & y_9 \arrow[r, "\text{Ber}"] & y_{10} \arrow[r, "\text{Samp}"] & y_{11} \arrow[r, "\text{Poi}"] & y_{12}
\end{tikzcd}
$$

In this graph the "rows" are years (2017, 2018, and 2019) and the "columns" are data within years: first tillers (y_1, y_5, and y_9), then panicles (y_2, y_6, and y_{10}), then sub-sampled panicles (y_3, y_7, and y_{11}), and finally (the terminal nodes) floret count (y_4, y_8, and y_{12}).

After initial analysis we may change some Poisson to negative binomial (a kind of over-dispersed Poisson), but first we see whether that seems necessary.

It is somewhat problematic that tiller counts for different years are for the same plant and these should be dependent. We allow for such dependence (somewhat) by putting individual effects in the model.

```r
pred <- c(0, 1, 2, 3, 0, 5, 6, 7, 0, 9, 10, 11)
fam <- rep(c(2, 1, 1, 2), times = 3)
fam
```

```
##  [1] 2 1 1 2 2 1 1 2 2 1 1 2
```

```r
vars <- unique(redata$varb)
vars
```

```
##  [1] "late17.tillers"           "late17.panicles"
##  [3] "late17.pans.sampled"      "late17.floret.count.total"
##  [5] "late18.tillers"           "late18.panicles"
```

```
##  [7] "late18.pans.sampled"       "late18.floret.count.total"
##  [9] "late19.tillers"            "late19.panicles"
## [11] "late19.pans.sampled"       "late19.floret.count.total"
```

```r
pred.names <- c("initial", vars)[pred + 1]
foo <- cbind(pred.names, vars, fam)
colnames(foo) <- c("predecessor", "successor", "family")
foo
```

```
##       predecessor           successor                   family
##  [1,] "initial"             "late17.tillers"            "2"
##  [2,] "late17.tillers"      "late17.panicles"           "1"
##  [3,] "late17.panicles"     "late17.pans.sampled"       "1"
##  [4,] "late17.pans.sampled" "late17.floret.count.total" "2"
##  [5,] "initial"             "late18.tillers"            "2"
##  [6,] "late18.tillers"      "late18.panicles"           "1"
##  [7,] "late18.panicles"     "late18.pans.sampled"       "1"
##  [8,] "late18.pans.sampled" "late18.floret.count.total" "2"
##  [9,] "initial"             "late19.tillers"            "2"
## [10,] "late19.tillers"      "late19.panicles"           "1"
## [11,] "late19.panicles"     "late19.pans.sampled"       "1"
## [12,] "late19.pans.sampled" "late19.floret.count.total" "2"
```

```r
fit <- as.numeric(grepl("floret", as.character(redata$varb)))
redata <- data.frame(redata, fit = fit)
ind <- as.factor(redata$id)
redata <- data.frame(redata, ind = ind)

redata <- subset(redata, redata$varb %in% vars)

nnode <- length(vars)
nind <- length(unique(redata$id))
nnode * nind == nrow(redata)
```

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

4 Initial Aster Models

4.1 Fit models (Poisson)

To start, we'll fit three models: a null model, a model with a fixed effect term for symptom extent (`symp.extent`) (a plant-level measure we assessed each growing season), and a third model that includes a term to describe variation at the individual level (`ind`).

```r
anull <- aster(resp ~ varb,
    pred, fam, varb, id, root, data = redata)

aout.noind <- aster(resp ~ varb + fit : (symp.extent),
    pred, fam, varb, id, root, data = redata)

aout <- aster(resp ~ varb + fit : (symp.extent + ind),
    pred, fam, varb, id, root, data = redata)



anova(anull,aout.noind,aout)
```

```
## Analysis of Deviance Table
##
## Model 1: resp ~ varb
## Model 2: resp ~ varb + fit:(symp.extent)
## Model 3: resp ~ varb + fit:(symp.extent + ind)
##   Model Df Model Dev Df Deviance P(>|Chi|)
## 1       12   1283304
## 2       13   1283318  1   14.419 0.0001463 ***
## 3       22   1283360  9   41.634 3.833e-06 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

The hypothesis test says both `symp.extent` and `ind` are statistically significant. The `ind` term helps us model the dependency of the tiller counts at the individual plant level, which were different among plants at the start of the experiment. Additionally, we also know (from other experiments) that there is a lot of variation in panicle lengths and floret production at the individual plant level. Therefore, going forward, we'll use the `ind` model as the base model as we evaluate residuals and over-dispersion.

```r
summary(aout, info.tol = 1e-9)
```

```
##
## Call:
## aster.formula(formula = resp ~ varb + fit:(symp.extent + ind),
##     pred = pred, fam = fam, varvar = varb, idvar = id, root = root,
##     data = redata)
##
##                                 Estimate Std. Error  z value Pr(>|z|)
## (Intercept)                    6.006e+00  5.120e-03 1172.906  < 2e-16 ***
## varblate17.panicles           -3.888e+00  7.796e-02  -49.870  < 2e-16 ***
## varblate17.pans.sampled       -4.124e+02  2.021e+00 -204.073  < 2e-16 ***
## varblate17.tillers            -2.292e+00  7.003e-02  -32.730  < 2e-16 ***
## varblate18.floret.count.total -3.922e-02  6.868e-03   -5.711 1.13e-08 ***
## varblate18.panicles           -4.008e+00  6.770e-02  -59.208  < 2e-16 ***
## varblate18.pans.sampled       -3.970e+02  1.852e+00 -214.407  < 2e-16 ***
## varblate18.tillers            -1.984e+00  6.009e-02  -33.017  < 2e-16 ***
## varblate19.floret.count.total -2.994e-01  6.916e-03  -43.283  < 2e-16 ***
## varblate19.panicles           -4.341e+00  5.463e-02  -79.463  < 2e-16 ***
## varblate19.pans.sampled       -3.078e+02  1.448e+00 -212.613  < 2e-16 ***
## varblate19.tillers            -1.471e+00  4.661e-02  -31.561  < 2e-16 ***
## fit:symp.extent               -8.189e-03  4.020e-03   -2.037 0.041653 *
## fit:ind2                       1.562e-03  8.528e-04    1.832 0.066964 .
## fit:ind3                       1.750e-03  7.745e-04    2.260 0.023826 *
## fit:ind4                       1.578e-03  1.771e-03    0.891 0.372840
## fit:ind5                       8.084e-04  7.792e-04    1.037 0.299554
## fit:ind6                      -1.225e-03  1.186e-03   -1.033 0.301691
## fit:ind7                       1.379e-03  8.850e-04    1.558 0.119230
## fit:ind8                      -8.420e-04  1.125e-03   -0.748 0.454169
## fit:ind9                       4.081e-04  8.167e-04    0.500 0.617302
## fit:ind10                      2.356e-03  6.985e-04    3.373 0.000744 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

4.2 Get Conditional and Unconditional Mean Value Parameters

```
pout.cond <- predict(aout, model.type = "conditional",
    is.always.parameter = TRUE, gradient = TRUE)
xi <- pout.cond$fit
class(xi)
```

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

```
length(xi) == nind * nnode
```

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

```
xi <- matrix(xi, nrow = nind)
colnames(xi) <- vars
xi
```

```
##       late17.tillers late17.panicles late17.pans.sampled
##  [1,]       109.6623       0.8130624          0.04443901
##  [2,]       113.1557       0.8188337          0.08046659
##  [3,]       114.5872       0.8210970          0.09445717
##  [4,]       108.8709       0.8117036          0.03588178
##  [5,]       117.4830       0.8255067          0.12149555
##  [6,]       113.9266       0.8200595          0.08805372
##  [7,]       123.2334       0.8336490          0.17066907
##  [8,]       115.6305       0.8227111          0.10438804
##  [9,]       114.7692       0.8213806          0.09620495
## [10,]       125.6813       0.8368890          0.18997002
##       late17.floret.count.total late18.tillers late18.panicles
##  [1,]                  404.3292       138.8843       0.7991133
##  [2,]                  404.9614       140.3235       0.8011737
##  [3,]                  405.1370       142.4651       0.8041626
##  [4,]                  404.1064       134.6663       0.7928213
##  [5,]                  405.4191       142.2782       0.8039053
##  [6,]                  405.0598       140.0113       0.8007304
##  [7,]                  405.8165       151.9228       0.8163542
##  [8,]                  405.2480       140.2137       0.8010180
##  [9,]                  405.1573       140.2947       0.8011329
## [10,]                  405.9472       149.9400       0.8139256
##       late18.pans.sampled late18.floret.count.total late19.tillers
##  [1,]          0.07374258                  389.5105       180.9998
##  [2,]          0.08560038                  389.6725       184.5408
##  [3,]          0.10269359                  389.8735       187.0294
##  [4,]          0.03714979                  388.7862       176.8308
##  [5,]          0.10122746                  389.8574       183.7563
##  [6,]          0.08305444                  389.6395       181.2938
##  [7,]          0.17112048                  390.4634       188.1139
##  [8,]          0.08470643                  389.6610       181.7812
##  [9,]          0.08536599                  389.6695       185.6416
## [10,]          0.15765326                  390.3653       191.0123
##       late19.panicles late19.pans.sampled late19.floret.count.total
##  [1,]       0.7425412          0.08407609                  300.3656
##  [2,]       0.7474814          0.10758842                  300.6382
##  [3,]       0.7508413          0.12340281                  300.7933
##  [4,]       0.7364712          0.05475488                  299.9053
##  [5,]       0.7464032          0.10248368                  300.5839
```

```
##  [6,]       0.7429587         0.08607541                   300.3913
##  [7,]       0.7522778         0.13012098                   300.8540
##  [8,]       0.7436478         0.08937021                   300.4325
##  [9,]       0.7489787         0.11465373                   300.7098
## [10,]       0.7560367         0.14757983                   301.0001
```

```r
pout.unco <- predict(aout, gradient = TRUE)
mu <- pout.unco$fit
mu <- matrix(mu, nrow = nind)
colnames(mu) <- vars
mu
```

```
##       late17.tillers late17.panicles late17.pans.sampled
##  [1,]       109.6623        89.16228            3.962284
##  [2,]       113.1557        92.65569            7.455688
##  [3,]       114.5872        94.08721            8.887212
##  [4,]       108.8709        88.37091            3.170905
##  [5,]       117.4830        96.98300           11.783003
##  [6,]       113.9266        93.42656            8.226556
##  [7,]       123.2334       102.73342           17.533416
##  [8,]       115.6305        95.13048            9.930485
##  [9,]       114.7692        94.26916            9.069160
## [10,]       125.6813       105.18129           19.981292
##       late17.floret.count.total late18.tillers late18.panicles
##  [1,]                  1602.067       138.8843        110.9843
##  [2,]                  3019.266       140.3235        112.4235
##  [3,]                  3600.538       142.4651        114.5651
##  [4,]                  1281.383       134.6663        106.7663
##  [5,]                  4777.054       142.2782        114.3782
##  [6,]                  3332.247       140.0113        112.1113
##  [7,]                  7115.350       151.9228        124.0228
##  [8,]                  4024.309       140.2137        112.3137
##  [9,]                  3674.436       140.2947        112.3947
## [10,]                  8111.349       149.9400        122.0400
##       late18.pans.sampled late18.floret.count.total late19.tillers
##  [1,]            8.184266                  3187.858       180.9998
##  [2,]            9.623494                  3750.011       184.5408
##  [3,]           11.765102                  4586.901       187.0294
##  [4,]            3.966348                  1542.061       176.8308
##  [5,]           11.578217                  4513.854       183.7563
##  [6,]            9.311345                  3628.068       181.2938
##  [7,]           21.222849                  8286.746       188.1139
##  [8,]            9.513691                  3707.115       181.7812
##  [9,]            9.594683                  3738.756       185.6416
## [10,]           19.240005                  7510.630       191.0123
##       late19.panicles late19.pans.sampled late19.floret.count.total
##  [1,]        134.3998           11.299811                  3394.075
##  [2,]        137.9408           14.840837                  4461.723
##  [3,]        140.4294           17.329381                  5212.561
##  [4,]        130.2308            7.130771                  2138.556
##  [5,]        137.1563           14.056280                  4225.092
##  [6,]        134.6938           11.593826                  3482.685
##  [7,]        141.5139           18.413932                  5539.904
##  [8,]        135.1812           12.081169                  3629.576
##  [9,]        139.0416           15.941643                  4793.808
```

```
## [10,]        144.4123           21.312350                    6415.020
```

4.3 Correct for Sub-sampling

4.3.1 Point Estimates

The fundamental relationship between conditional and unconditional means is

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

So we get unconditional means by multiplying together the corresponding conditional mean and the unconditional mean for the predecessor.

To correct for sub-sampling, we want to do the same thing except we want to leave out the sub-sampling arrows. That is

$$\mu_{\text{florets}} = \mu_{\text{panicles}} \xi_{\text{florets}}$$

So first we obtain these quantities.

```
is.florets <- grep("floret", vars)
is.panicles <- grep("panicles", vars)
is.florets
```

```
## [1]  4  8 12
```

```
is.panicles
```

```
## [1]  2  6 10
```

```
mu.panicles <- mu[ , is.panicles]
xi.florets <- xi[ , is.florets]
mu.florets <- mu.panicles * xi.florets
mu.florets
```

```
##       late17.panicles late18.panicles late19.panicles
##  [1,]        36050.92        43229.54        40369.09
##  [2,]        37521.98        43808.35        41470.29
##  [3,]        38118.21        44665.89        42240.21
##  [4,]        35711.25        41509.28        39056.89
##  [5,]        39318.76        44591.20        41226.97
##  [6,]        37843.34        43683.01        40460.86
##  [7,]        41690.92        48426.38        42575.03
##  [8,]        38551.44        43764.27        40612.82
##  [9,]        38193.84        43796.78        41811.18
## [10,]        42698.05        47640.18        43468.14
```

Then (the best surrogate of) fitness (in these data) is the sum of these for each individual.

```
mu.fit <- rowSums(mu.florets)
mu.fit
```

```
##  [1] 119649.5 122800.6 125024.3 116277.4 125136.9 121987.2 132692.3 122928.5
##  [9] 123801.8 133806.4
```

4.4 Standard Errors

For reasons that will soon become apparent, we make an R function to do the preceding calculation.

```
foo <- function(x) {
    # x is xi and mu strung out as one vector
```

```
    xi <- x[1:length(xi)]
    mu <- x[- (1:length(xi))]
    xi <- matrix(xi, nrow = nind)
    mu <- matrix(mu, nrow = nind)
    mu.panicles <- mu[ , is.panicles]
    xi.florets <- xi[ , is.florets]
    mu.florets <- mu.panicles * xi.florets
    mu.fit <- rowSums(mu.florets)
}
```

And we check that it does indeed give the same calculation as above.

```
ximu <- c(xi, mu)
all.equal(foo(ximu), mu.fit)
```

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

In order to derive standard errors using the delta method, we need Jacobian matrices (matrices of partial derivatives). Rather than do any calculus, we let R package `numDeriv` figure out the Jacobian matrix for this transformation. We also need the Jacobian matrix for the transformation from the "coefficients" vector to the vector `ximu`.

```
jac.foo <- jacobian(foo, ximu)
jac.ximu <- rbind(pout.cond$gradient, pout.unco$gradient)
```

Now the chain rule from multivariate calculus says the Jacobian for the overall transformation is the product of the Jacobians for the parts.

```
jac.total <- jac.foo %*% jac.ximu
```

Now the delta method says the variance-covariance matrix of all the fitnesses (the vector estimate `mu.fit`) is $J I^{-1} J^T$, where J is the overall Jacobian matrix `jac.total` and I is Fisher information for the "coefficients" vector

```
V <- jac.total %*% solve(aout$fisher) %*% t(jac.total)
```

and the standard errors are square roots of the variances (the diagonal elements of `V`)

```
se <- sqrt(diag(V))
bar.pois <- cbind(mu.fit, se)
colnames(bar.pois) <- c("Estimate", "SE")
```

Table 1: Estimated Fitness with Standard Error for Different Individuals (Poisson Distributions for Tillers and Florets)

Estimate	SE
119649.5	2702.672
122800.6	2903.538
125024.3	3037.413
116277.4	2478.442
125136.9	3062.308
121987.2	2865.781
132692.3	3495.365
122928.5	2927.697
123801.8	2965.634
133806.4	3547.607

5 Checking for Over-dispersion

Following the theory for the negative binomial distribution, if the conditional mean value parameter is ξ and the shape parameter is α and the data are y, then the conditional variance is

$$\xi \left(1 + \frac{\xi}{\alpha} \right)$$

We use this to estimate the shape parameter. Let A be a set of nodes all of which we think might be negative binomial with the same shape parameter, and let $\hat{\xi}$ be the estimated conditional mean value parameter vector assuming the Poisson distribution. Then we equate empirical conditional variance with the formula above

$$\sum_{j \in A} (y_j - y_{p(j)} \hat{\xi}_j)^2 = \sum_{j \in A} y_{p(j)} \hat{\xi}_j \left(1 + \frac{\hat{\xi}_j}{\alpha} \right) \tag{$*$}$$

where $p(j)$ is the predecessor of j. The right-hand side is a decreasing function of α and has infimum

$$\sum_{j \in A} y_{p(j)} \hat{\xi}_j \tag{$**$}$$

So long as the left-hand side of $(*)$ is greater than $(**)$ there will be a unique solution for α. Otherwise there is no solution, in which case the y_j values are under-dispersed rather than over-dispersed, and negative binomial is not appropriate.

5.1 Tillers

5.1.1 Observed and Conditional Mean Values

We assume the three arrows to the tillers nodes have the same over-dispersion. For these arrows, the predecessor is the constant 1. We have $y_{p(j)} = 1$ in $(*)$ and $(**)$.

```r
is.tiller <- grepl("tiller", redata$varb)
y.tiller <- redata$resp[is.tiller]
xi.tiller <- xi[is.tiller]
```

5.1.2 Pearson Residuals

So-called Pearson residuals are deviations from the (estimated) mean divided by the (estimated) standard error. For Poisson (which we used for the fitted model we are diagnosing now) the standard deviation is the square root of the mean, hence

```r
resid.pois.t <- (y.tiller - xi.tiller) / sqrt(xi.tiller)
stem(resid.pois.t, scale = 2)
```

```
##
##   The decimal point is at the |
##
##   -7 | 0
##   -6 | 554
##   -5 | 1
##   -4 | 700
##   -3 | 953
##   -2 | 9430
##   -1 |
##   -0 | 44
##    0 | 0
##    1 |
```

```
##    2 | 28
##    3 | 588
##    4 | 47
##    5 |
##    6 |
##    7 | 124
##    8 | 34
```

```r
resid.pois.tills <- stem(resid.pois.t, scale = 2)
```

```
##
##   The decimal point is at the |
##
##   -7 | 0
##   -6 | 554
##   -5 | 1
##   -4 | 700
##   -3 | 953
##   -2 | 9430
##   -1 |
##   -0 | 44
##    0 | 0
##    1 |
##    2 | 28
##    3 | 588
##    4 | 47
##    5 |
##    6 |
##    7 | 124
##    8 | 34
```

We do not expect such large residuals in such a small sample. Thus we think we need negative binomial.

5.1.3 Estimating Shape Parameter for Tillers

```r
lhs <- sum((y.tiller - xi.tiller)^2)
rhs.min <- sum(xi.tiller)
lhs > rhs.min
```

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

Thus we can fit negative binomial. Write a function the zero of which is our estimate of the shape parameter.

```r
baz <- function(alpha) lhs - sum(xi.tiller * (1 + xi.tiller / alpha))
```

Then we find two points where this function has opposite signs and feed it to R function `uniroot`.

```r
baz(1)
```

```
## [1] -573518.5
```

```r
baz(10)
```

```
## [1] 34305.9
```

```r
uout <- uniroot(baz, c(1, 10), tol = sqrt(.Machine$double.eps))
uout
```

```
## $root
```

```
## [1] 6.631457
##
## $f.root
## [1] 0
##
## $iter
## [1] 8
##
## $init.it
## [1] NA
##
## $estim.prec
## [1] 0.002350784
```

Looks like we want negative binomial with shape parameter 6.6314567 for this first arrow.

```r
famlist <- list(fam.bernoulli(), fam.poisson(),
    fam.negative.binomial(uout$root))
# assign the nb to all tiller nodes
fam[grep("tillers", vars)] <- 3
famlist
```

```
## [[1]]
## [1] "bernoulli"
##
## [[2]]
## [1] "poisson"
##
## [[3]]
## [1] "negative.binomial(size = 6.63145669255945)"
```

5.1.4 Model fitting

Now do everything all over again, and check for over-dispersion for florets.

```r
aout <- aster(resp ~ varb + fit : (symp.extent + ind),
    pred, fam, varb, id, root, data = redata, famlist = famlist)
```

```
## Warning in aster.default(x, root, pred, fam, modmat, parm, type, famlist, :
## Algorithm did not converge
```

To avoid convergence trouble, let's bump up the max iterations and then see if convergence comes more easily after we finalize the shape parameter estimates.

```r
aout <- aster(resp ~ varb + fit : (symp.extent + ind),
    pred, fam, varb, id, root, data = redata, famlist = famlist, maxiter = 20000)
```

5.2 Florets

Now we look at terminal arrows, using the same shape parameter for all years.

5.2.1 Observed, Predecessors, and Conditional Mean Values

```r
is.floret <- grep("floret.count.total", redata$varb)
is.floret.pred <- grep("pans.sampled", redata$varb)
y.floret <- redata$resp[is.floret]
```

```r
y.floret.pred <- redata$resp[is.floret.pred]
xi.floret <- xi[is.floret]
```

5.2.2 Pearson Residuals

```r
resid.pois.f <- (y.floret - y.floret.pred * xi.floret) / sqrt(y.floret.pred * xi.floret)
summary(resid.pois.f)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
## -36.646  -9.606   1.307   1.047  10.897  72.895
```

```r
stem(resid.pois.f, scale = 2)
```

```
##
##   The decimal point is 1 digit(s) to the right of the |
##
##   -3 | 70
##   -2 | 5
##   -1 | 86550
##   -0 | 988754
##    0 | 1145799
##    1 | 013347
##    2 | 08
##    3 |
##    4 |
##    5 |
##    6 |
##    7 | 3
```

```r
resid.pois.florets <- stem(resid.pois.f, scale = 2)
```

```
##
##   The decimal point is 1 digit(s) to the right of the |
##
##   -3 | 70
##   -2 | 5
##   -1 | 86550
##   -0 | 988754
##    0 | 1145799
##    1 | 013347
##    2 | 08
##    3 |
##    4 |
##    5 |
##    6 |
##    7 | 3
```

We do not expect such large residuals (more than 4 standard deviations from the mean) in such a small sample. Thus we think we need negative binomial.

5.2.3 Estimating Shape Parameter for Florets

```r
lhs.f <- sum((y.floret - y.floret.pred * xi.floret)^2)
rhs.min.f <- sum(y.floret.pred * xi.floret)
lhs.f > rhs.min.f
```

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

Thus we can fit negative binomial. Write a function the zero of which is our estimate of the shape parameter.

```
baz.f <- function(alpha) lhs.f -
    sum(y.floret.pred * xi.floret * (1 + xi.floret / alpha))
```

Then we find two points where this function has opposite signs and feed it to R function `uniroot`.

```
baz.f(1)
```

```
## [1] -4097496
```

```
baz.f(10)
```

```
## [1] 37998248
```

```
uout.f <- uniroot(baz.f, c(1, 10), tol = sqrt(.Machine$double.eps))
uout.f
```

```
## $root
## [1] 1.096015
##
## $f.root
## [1] 0.0002074242
##
## $iter
## [1] 7
##
## $init.it
## [1] NA
##
## $estim.prec
## [1] 7.450581e-09
```

Looks like we want negative binomial with shape parameter 1.0960151 for these terminal arrows.

```
famlist <- c(famlist, list(fam.negative.binomial(uout.f$root)))
famlist
```

```
## [[1]]
## [1] "bernoulli"
##
## [[2]]
## [1] "poisson"
##
## [[3]]
## [1] "negative.binomial(size = 6.63145669255945)"
##
## [[4]]
## [1] "negative.binomial(size = 1.09601506363191)"
```

```
fam[grep("floret", vars)] <- 4
fam
```

```
##  [1] 3 1 1 4 3 1 1 4 3 1 1 4
```

5.2.4 Model Fitting

Fit model with NB distributions for tillers and florets

```
aout <- aster(resp ~ varb + fit : (symp.extent + ind),
    pred, fam, varb, id, root, data = redata, famlist = famlist)
```

5.3 Redo Conditional and Unconditional Mean Value Parameters

```
pout.cond <- predict(aout, model.type = "conditional",
     is.always.parameter = TRUE, gradient = TRUE)
xi <- pout.cond$fit
pout.unco <- predict(aout, gradient = TRUE)
mu <- pout.unco$fit
ximu <- c(xi, mu)
```

5.4 Redo Jacobian matrices

```
jac.foo <- jacobian(foo, ximu)
jac.ximu <- rbind(pout.cond$gradient, pout.unco$gradient)
jac.total <- jac.foo %*% jac.ximu
```

5.5 Estimate fitness

Re-do the delta method to estimate fitness.

```
V <- jac.total %*% solve(aout$fisher) %*% t(jac.total)

se <- sqrt(diag(V))
bar.nb1 <- cbind(foo(ximu), se)
colnames(bar.nb1) <- c("Estimate", "SE")
```

Table 2: Estimated Fitness with Standard Error for Different Individuals (Initial Negative Binomial Distributions for Tillers and Florets)

Estimate	SE
89147.71	23208.04
112386.97	28338.03
128489.18	32061.15
62850.23	17022.06
128860.78	31942.63
106362.79	26877.86
181763.34	44533.71
113017.06	28360.42
119496.72	29896.84
189029.06	45572.24

6 Re-estimating over-dispersion for final model

Now that we have fit the `ind` model with two negative binomial distributions, we

- re-estimate ξ,
- re-estimate the negative binomial shape parameters based on this new $\hat{\xi}$.

And we do this repeatedly until the estimates of shape parameters converge. We'll use the initial estimates for the NB shape parameters as the starting point for the model.

```
shapes.save <- lapply(famlist, function(x) x$size)
shapes.save <- unlist(shapes.save)

for (i in 1:7) {
xi <- predict(aout, model.type = "conditional", is.always.parameter = TRUE)

# tillers
xi.tiller <- xi[is.tiller]

lhs <- sum((y.tiller - xi.tiller)^2)
rhs.min <- sum(xi.tiller)
stopifnot(lhs > rhs.min)

baz <- function(alpha) lhs - sum(xi.tiller * (1 + xi.tiller / alpha))
uout <- uniroot(baz, c(2, 20), tol = sqrt(.Machine$double.eps),
    extendInt = "yes")
famlist[[3]] <- fam.negative.binomial(uout$root)

# florets
xi.floret <- xi[is.floret]

lhs <- sum((y.floret - y.floret.pred * xi.floret)^2)
rhs.min <- sum(y.floret.pred * xi.floret)
stopifnot(lhs > rhs.min)

baz <- function(alpha) lhs -
    sum(y.floret.pred * xi.floret * (1 + xi.floret / alpha))
uout <- uniroot(baz, c(1/2, 2), tol = sqrt(.Machine$double.eps),
    extendInt = "yes")
famlist[[4]] <- fam.negative.binomial(uout$root)

aout <- aster(resp ~ varb + fit : (symp.extent + ind),
    pred, fam, varb, id, root, data = redata, famlist = famlist,
    maxiter = 20000)

shapes.tmp <- lapply(famlist, function(x) x$size)
shapes.save <- rbind(shapes.save, unlist(shapes.tmp))
}
rownames(shapes.save) <- NULL
colnames(shapes.save) <- c("tillers", "florets")
```

Let's take a look at where the shape parameters converged for the negative binomial model.

Table 3: Estimated shape parameters of negative binomial distributions, each row one iteration

tillers	florets
6.6315	1.0960
11.9381	1.2094
10.1647	1.1872
10.6408	1.1950

tillers	florets
10.5046	1.1930
10.5430	1.1936
10.5322	1.1935
10.5352	1.1935

The initial NB shape parameter for tillers was 6.631, and it converged at 10.535. The change in the shape parameter for florets was less drastic. Initially, it was 1.096, and it converged at 1.193.

7 Evaluating final model

The famlist was automatically updated during the convergence process, so we just need to rerun the null and final model in order to compare them.

```
anull <- aster(resp ~ varb,pred, fam, varb, id, root, data = redata, famlist = famlist)

aout <- aster(resp ~ varb + fit : (symp.extent + ind),
    pred, fam, varb, id, root, data = redata, famlist = famlist)

anova(anull,aout)
```

```
## Analysis of Deviance Table
##
## Model 1: resp ~ varb
## Model 2: resp ~ varb + fit:(symp.extent + ind)
##   Model Df Model Dev Df Deviance P(>|Chi|)
## 1       12    -14909
## 2       22    -14889 10   19.719   0.03203 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

With the final shape parameters for tillers and florets for the ind model, we didn't have any convergence trouble. We also see that our final model is still explaining a significant amount of variation relative the null.

8 Final fitness estimates

8.1 Get Conditional and Unconditional Mean Value Parameters

```
pout.cond.f <- predict(aout, model.type = "conditional",
     is.always.parameter = TRUE, gradient = TRUE)
xi <- pout.cond.f$fit
class(xi)
```

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

```
length(xi) == nind * nnode
```

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

```
xi <- matrix(xi, nrow = nind)
colnames(xi) <- vars
xi
```

```
##       late17.tillers late17.panicles late17.pans.sampled
```

```
##  [1,]      90.67461      0.8187365          0.07986425
##  [2,]     105.28175      0.8213544          0.09604337
##  [3,]     110.67019      0.8221456          0.10091307
##  [4,]      85.22798      0.8175307          0.07237712
##  [5,]     124.38953      0.8238506          0.11137532
##  [6,]     108.97500      0.8219051          0.09943399
##  [7,]     145.55836      0.8258509          0.12359413
##  [8,]     116.43644      0.8229112          0.10561608
##  [9,]     111.89728      0.8223151          0.10195526
## [10,]     157.88888      0.8267688          0.12918153
##       late17.floret.count.total late18.tillers late18.panicles
##  [1,]                  307.3460       125.2207       0.8018327
##  [2,]                  364.3105       133.3189       0.8027668
##  [3,]                  381.5037       143.5432       0.8037956
##  [4,]                  280.9967       102.9123       0.7984990
##  [5,]                  418.5753       143.7408       0.8038141
##  [6,]                  376.2781       132.4406       0.8026710
##  [7,]                  462.1763       194.4716       0.8073089
##  [8,]                  398.1430       133.5357       0.8027903
##  [9,]                  385.1878       133.0589       0.8027386
## [10,]                  482.2485       178.7573       0.8064384
##       late18.pans.sampled late18.floret.count.total late19.tillers
##  [1,]          0.08937983                  346.1512       163.5410
##  [2,]          0.09472708                  365.2839       188.0090
##  [3,]          0.10060202                  386.3426       205.2266
##  [4,]          0.07019538                  277.5825       135.8233
##  [5,]          0.10070720                  386.7200       182.9424
##  [6,]          0.09417931                  363.3227       167.5384
##  [7,]          0.12055089                  458.3223       205.0083
##  [8,]          0.09486116                  365.7640       170.6834
##  [9,]          0.09456566                  364.7059       195.7690
## [10,]          0.11562430                  440.4627       226.4587
##       late19.panicles late19.pans.sampled late19.floret.count.total
##  [1,]       0.7451543          0.09655191                  276.6657
##  [2,]       0.7471616          0.10607578                  302.1346
##  [3,]       0.7482871          0.11139389                  316.4095
##  [4,]       0.7420068          0.08151413                  236.6059
##  [5,]       0.7467900          0.10431668                  297.4219
##  [6,]       0.7455223          0.09830178                  281.3375
##  [7,]       0.7482740          0.11133216                  316.2436
##  [8,]       0.7457997          0.09961978                  284.8583
##  [9,]       0.7476934          0.10859044                  308.8791
## [10,]       0.7494395          0.11682196                  331.0277
```

```r
pout.unco.f <- predict(aout, gradient = TRUE)
mu <- pout.unco.f$fit
mu <- matrix(mu, nrow = nind)
colnames(mu) <- vars
mu
```

```
##       late17.tillers late17.panicles late17.pans.sampled
##  [1,]       90.67461        74.23861            5.929011
##  [2,]      105.28175        86.47362            8.305218
##  [3,]      110.67019        90.98700            9.181778
##  [4,]       85.22798        69.67649            5.042984
```

```
##  [5,]      124.38953       102.47839           11.413563
##  [6,]      108.97500        89.56711            8.906015
##  [7,]      145.55836       120.20950           14.857189
##  [8,]      116.43644        95.81684           10.119799
##  [9,]      111.89728        92.01482            9.381395
## [10,]      157.88888       130.53761           16.863048
##       late17.floret.count.total late18.tillers late18.panicles
##  [1,]                  1822.258       125.2207       100.40607
##  [2,]                  3025.678       133.3189       107.02402
##  [3,]                  3502.882       143.5432       115.37937
##  [4,]                  1417.062       102.9123        82.17538
##  [5,]                  4777.436       143.7408       115.54086
##  [6,]                  3351.138       132.4406       106.30625
##  [7,]                  6866.640       194.4716       156.99862
##  [8,]                  4029.127       133.5357       107.20119
##  [9,]                  3613.599       133.0589       106.81150
## [10,]                  8132.179       178.7573       144.15673
##       late18.pans.sampled late18.floret.count.total late19.tillers
##  [1,]            8.974277                  3106.457       163.5410
##  [2,]           10.138073                  3703.275       188.0090
##  [3,]           11.607398                  4484.432       205.2266
##  [4,]            5.768332                  1601.188       135.8233
##  [5,]           11.635796                  4499.795       182.9424
##  [6,]           10.011850                  3637.532       167.5384
##  [7,]           18.926324                  8674.356       205.0083
##  [8,]           10.169230                  3719.538       170.6834
##  [9,]           10.100700                  3683.785       195.7690
## [10,]           16.668020                  7341.641       226.4587
##       late19.panicles late19.pans.sampled late19.floret.count.total
##  [1,]        121.8633           11.766132                  3255.285
##  [2,]        140.4731           14.900797                  4502.047
##  [3,]        153.5684           17.106582                  5412.685
##  [4,]        100.7818            8.215139                  1943.750
##  [5,]        136.6196           14.251701                  4238.768
##  [6,]        124.9036           12.278243                  3454.330
##  [7,]        153.4024           17.078621                  5401.004
##  [8,]        127.2956           12.681160                  3612.334
##  [9,]        146.3752           15.894944                  4909.616
## [10,]        169.7171           19.826680                  6563.180
```

8.2 Correcting for Sub-sampling

8.2.1 Point Estimates

```r
is.florets <- grep("floret", vars)
is.panicles <- grep("panicles", vars)
is.florets
```

```
## [1]  4  8 12
```

```r
is.panicles
```

```
## [1]  2  6 10
```

```r
mu.panicles <- mu[ , is.panicles]
xi.florets <- xi[ , is.florets]
```

```r
mu.florets <- mu.panicles * xi.florets
mu.florets
```

```
##       late17.panicles late18.panicles late19.panicles
##  [1,]        22816.94        34755.68        33715.39
##  [2,]        31503.25        39094.15        42441.80
##  [3,]        34711.88        44575.96        48590.51
##  [4,]        19578.86        22810.45        23845.56
##  [5,]        42894.93        44681.97        40633.66
##  [6,]        33702.14        38623.47        35140.06
##  [7,]        55557.98        71955.96        48512.53
##  [8,]        38148.81        39210.34        36261.22
##  [9,]        35442.99        38954.79        45212.23
## [10,]        62951.56        63495.66        56181.05
```

Now calculate the final fitness estimates.

```r
mu.fit <- rowSums(mu.florets)
mu.fit
```

```
##  [1]  91288.01 113039.20 127878.35  66234.87 128210.55 107465.67 176026.47
##  [8] 113620.36 119610.00 182628.26
```

Check

```r
ximu <- c(xi, mu)
all.equal(foo(ximu), mu.fit)
```

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

Final Jacobian matrices

```r
jac.foo <- jacobian(foo, ximu)
jac.ximu <- rbind(pout.cond.f$gradient, pout.unco.f$gradient)
jac.total <- jac.foo %*% jac.ximu
```

Delta method for final fitness estimates.

```r
V <- jac.total %*% solve(aout$fisher) %*% t(jac.total)
```

The standard errors are square roots of the variances (the diagonal elements of `V`)

```r
se.final <- sqrt(diag(V))
bar.final <- cbind(mu.fit, se.final)
colnames(bar.final) <- c("Estimate", "SE")
```

Table 4: Estimated Fitness with Standard Error for Different Individuals (Final Negative Binomial Distributions for Tillers and Florets

Estimate	SE
91288.01	19816.16
113039.20	23636.53
127878.35	26353.40
66234.88	15136.14
128210.55	26297.39
107465.66	22584.03
176026.47	35303.11
113620.36	23671.55

Estimate	SE
119610.00	24776.94
182628.26	36099.52

Caution: Standard errors involving negative binomial arrows do not account for estimating the shape parameters of these negative binomial distributions. Whenever such are presented, some academic weasel wording must be emitted to refer to this fact. More precisely, the standard errors in Table 3 assume the size parameters of the negative binomial distributions are known rather than estimated. They do correctly account for sampling variability under that assumption, asymptotically (for sufficiently large sample size).

9 Comparing Pearson residuals for models

```
is.tiller <- grepl("tiller", redata$varb)
y.tiller <- redata$resp[is.tiller]
xi.tiller <- xi[is.tiller]
resid.t.final <- (y.tiller - xi.tiller) /
    sqrt(xi.tiller * (1 + xi.tiller / famlist[[3]]$size))
summary(resid.t.final)
```

```
##     Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
## -1.77192 -0.87328 -0.24797 -0.07649  0.72515  2.14036
```

```
stem(resid.t.final, scale = 1)
```

```
##
##   The decimal point is at the |
##
##   -1 | 876
##   -1 | 321
##   -0 | 997765
##   -0 | 4432
##    0 | 12
##    0 | 666778
##    1 | 11122
##    1 |
##    2 | 1
```

Compare these to the Pearson residuals for tillers from the initial Poisson model.

```
stem(resid.pois.t, scale = 2)
```

```
##
##   The decimal point is at the |
##
##   -7 | 0
##   -6 | 554
##   -5 | 1
##   -4 | 700
##   -3 | 953
##   -2 | 9430
##   -1 |
##   -0 | 44
##    0 | 0
##    1 |
```

```
##    2 | 28
##    3 | 588
##    4 | 47
##    5 |
##    6 |
##    7 | 124
##    8 | 34
```

The residual analysis for the original model (with Poisson arrows for tillers and florets) clearly showed the model did not fit the data because the residuals were far larger than standard normal (which they would be only for very large sample sizes, which we do not have here, but still the residuals are far larger than they should be). The residual analysis for the final model (with negative binomial arrows for tillers and florets) shows no lack of fit of the model data because the residuals are the same size as standard normal residuals would be, although not quite standard normal in distribution, perhaps. But there isn't anywhere else in aster models to go. So we declare this model fits and move on (at least as far as tillers are concerned).

On to florets.

```r
resid.f.final <-
    (y.floret - y.floret.pred * xi.floret) /
    sqrt(y.floret.pred * xi.floret * (1 + xi.floret / famlist[[4]]$size))
summary(resid.f.final)
```

```
##      Min.   1st Qu.    Median      Mean   3rd Qu.      Max.
## -2.164408 -0.654005 -0.006392  0.134414  0.738515  4.545859
```

```r
stem(resid.f.final, scale = 2)
```

```
##
##   The decimal point is at the |
##
##   -2 | 2
##   -1 | 5
##   -1 | 2
##   -0 | 9887755
##   -0 | 444300
##    0 | 333
##    0 | 577889
##    1 | 234
##    1 | 5
##    2 |
##    2 |
##    3 |
##    3 |
##    4 |
##    4 | 5
```

Compare these to initial residuals from Poisson model

```r
stem(resid.pois.f, scale = 2)
```

```
##
##   The decimal point is 1 digit(s) to the right of the |
##
##   -3 | 70
##   -2 | 5
##   -1 | 86550
##   -0 | 988754
```

```
##     0 | 1145799
##     1 | 013347
##     2 | 08
##     3 |
##     4 |
##     5 |
##     6 |
##     7 | 3
```

We still have the one outlier. Clearly that one observation does not fit either Poisson or negative binomial model. But the final model shows no other issues. The residuals are (except for the outlier) about the same size as standard normal residuals.